Partage et réutilisation de règles sur le Web de données

Oumy Seye

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION

ET DE LA COMMUNICATION

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice Sophia Antipolis et

de l'Université Gaston Berger de Saint-Louis Sénégal

Mention : INFORMATIQUE

Présentée et soutenue publiquement par

Oumy SEYE

le 15 décembre 2014

Partage et réutilisation de règles pour le Web de données

Jury :

Rapporteurs :	Sylvie DESPRÈS	Université Paris 13
	Ollivier HAEMMERLÉ	Université Toulouse Jean Jaurès
Directeurs :	Olivier CORBY	INRIA
	Fabien GANDON	INRIA
	Moussa LO	Université Gaston Berger
Encadrante :	Catherine FARON ZUCKER	Université Nice Sophia Antipolis
Examinateurs :	Chantal REYNAUD	Université Paris-Sud
	Cheikh Talibouya Diop	Université Gaston Berger
	Nhan Le Thanh	Université Nice Sophia Antipolis

A feu ma mère

A ma fille

A Dieng-Kuntz

REMERCIEMENTS

Je remercie Fabien Gandon directeur de l'équipe WIMMICS et Moussa Lo, Professeur à l'Université Gaston Berger de Saint-Louis pour m'avoir accordé leur confiance et le soutien financier à travers l'allocation Rose Dieng-Kuntz qui m'ont permis de faire cette thèse.

Mes remerciements vont particulièrement à mes encadrants, Catherine Faron-Zucker, maitre de conférence à L'Université de Nice et Olivier Corby, chargé de recherche à l'INRIA pour l'aide précieuse qu'ils m'ont apportée tout au long de cette thèse. Sans leur aide, le travail présenté dans cette thèse n'existerait pas. Je n'ai pas les mots pour leur témoigner ma gratitude, merci.

Je remercie Sylvie Després, Professeur à l'Université Paris 13 et Olivier Haemmerlé Professeur à l'Université Toulouse Jean Jaurès qui m'ont fait l'honneur d'être les rapporteurs de cette thèse. J'adresse mes remerciements à Chantal Reynault Professeur à l'Université Paris-Sud , Cheikh Talibouya Diop, maitre de conférence à l'Université Gaston Berger de Saint-Louis et Nhan LE THANH professeur à l'Université de Nice d'avoir accepté de participer à mon jury de thèse.

J'adresse mes remerciements à tous les membres de WIMMICS particulièrement mon camarade de bureau Nguyen THI HOA HUE pour avoir rendu agréable ce séjour de par leur bonne humeur quotidienne, mention spéciale à Alban Gainard et Maxime Lefrançois pour leur collaboration et leur aide. . Je remercie également tous les collègues de l'Université Gaston Berger de Saint Louis.

Enfin je remercie toute ma famille mon mari Mamadou pour son soutien, Thierno Seye, Fallou, Mor, Sohibou, Saliou,Dame, ma jumelle de cœur Adja Séne, mention spéciale à ma tante Astou Hanne pour son soutien moral et financier depuis la décès de ma maman, elle n'a ménagé aucun effort pour ma réussite.

Résumé : Dans cette thèse nous nous intéressons à la publication, au partage et à la réutilisation de règles sur le Web de données. L'approche que nous avons adoptée pour permettre la réutilisation de règles sur le Web, est de considérer des bases de règles comme des sources de données particulières. Il s'agit de les publier dans le langage RDF, le standard de représentation du Web de données. Nous utilisons des requêtes SPARQL, le standard du Web de données, pour interroger ces données RDF particulières qui représentent des règles. Nous proposons une traduction d'un sous-ensemble du langage RIF, le standard pour l'échange de règles sur le Web, en un sous-ensemble du langage SPARQL (les requêtes de la forme CONSTRUCT). Nous utilisons ensuite le langage SPIN pour traduire en RDF ces représentations de règles dans le langage SPARQL.

En d'autres termes, pour répondre au problème de la publication et la réutilisation de règles sur le Web, nous l'envisageons comme un problème classique en ingénierie des connaissances de partage et de réutilisation de connaissances. Nous proposons une approche basée sur (1) la représentation en RDF à la fois du contenu et des méta-données associées aux règles, (2) l'interopérabilité de cette représentation avec la recommandation RIF du W3C, (3) leur publication sur le Web de données et (4) la réutilisation de ces règles basée sur l'interrogation de sources de données RDF représentant des règles à l'aide de requêtes SPARQL. Nous avons construit un ensemble de requêtes SPARQL qui permettent (1) la construction de bases de règles spécifiques à un contexte ou une application particuliers, (2) l'optimisation des raisonnements par la sélection des seules règles pertinentes pour un jeu de données, (3) la validation de bases de règles par rapport à une source de données RDF et (4) la mise à jour de bases de règles.

L'implémentation et l'évaluation de nos travaux a été réalisée avec le moteur sémantique Corese/KGRAM.

Mots clés : Web de données liées, règles, SPARQL, SPIN, RIF

Abstract : In this thesis we address the problem of publishing, sharing and reusing rules on the Web of data. The approach adopted for sharing rules on the Web is to consider rule bases as particular data sources. Thus, we propose to publish rules using RDF, the standard representation language on the Web of data. We use the standard SPARQL language to query these particular RDF data that represent rules. We propose a translation of a subset of RIF (Rule Interchange Format), W3C standard for the exchange of rules on the Web, into a subset of SPARQL queries. Then we use the SPIN language for translating these SPARQL representations of rules into RDF.

In other words, we consider the problem of publishing and reusing rules on the Web as a classical problem in knowledge engineering : sharing and reusing knowledge. We propose an approach based on (1) the representation in RDF of rule content and metadata, (2) the interoperability of this representation with the W3C recommendation RIF, (3) the publication of rules on the Web of data and (4) reusing rules by querying RDF data sources representing them with the SPARQL query language. We built a set of SPARQL queries enabling (1) to build specific rule bases for a given context or application, (2) to optimize inference engines based on rule selection with respect to target RDF data sources, (3) to validate and update rule bases. We use the Corese/KGRAM semantic engine to implement and evaluate our proposals.

Keywords : Linked Open Data, Rules, SPARQL, SPIN, RIF

Table des matières

Liste des tableaux

Table des figures

CHAPITRE 1

Introduction

Contexte et motivations

Le partage et la réutilisation de connaissances sont les objectifs principaux du Web de données liées. Un ensemble de modèles, RDF, RDFS, OWL et SPARQL, ont été développés pour permettre aux humains et aux agents logiciels de publier et accéder aux données. Les schémas RDFS et les ontologies OWL constituent les premiers standards pour représenter la sémantique des connaissances d'un domaine sur le Web de données. La définition de règles d'inférences constitue un moyen complémentaire ou alternatif pour capturer la sémantique des connaissances sur le Web de données. Avec la standardisation du langage de règles RIF, la pile des standards du W3C propose une alternative pour la représentation et l'exploitation de la sémantique des données du Web.

Plusieurs autres langages de règles compatibles avec le Web sémantique ont été créés parallèlement à cette standardisation. Ainsi, les règles jouent de plus en plus un rôle important dans des applications du Web sémantique. En complément de données RDF, de schémas RDFS ou d'ontologies OWL, des règles peuvent être publiées et partagées sur le Web. Le problème qui se pose alors est de fournir un moyen de rechercher facilement des règles pertinentes, compatibles avec un jeu de données, pour une application particulière. La réutilisation de règles partagées pour une application spécifique nécessite de sélectionner parmi les règles partagées celles pertinentes pour les données de l'application. Cette sélection est la clé de la réussite de la réutilisation de règles pour leur adaptation à de nouveaux usages.

Contributions

Dans cette thèse nous nous intéressons au problème de la publication, du partage et de la réutilisation de règles sur le Web de données. Nous proposons une approche qui consiste à considérer des bases de règles comme des sources de données particulières. Il s'agit donc de les publier dans le langage RDF, le standard de représentation du Web de données. Nous utilisons des requêtes SPARQL, le standard du Web de données pour interroger ces données RDF particulières qui représentent des règles. Nous proposons une traduction d'un sous-ensemble du langage RIF, le standard pour l'échange de règles sur le Web, en un sous-ensemble du langage SPARQL. En d'autres termes, pour répondre au problème de la publication et la réutilisation de règles sur le Web, nous l'envisageons comme un problème classique en ingénierie des connaissances de partage et de réutilisation de connaissances et nous proposons une approche basée sur (1) la représentation en RDF à la fois du contenu et des méta-données associées aux règles, (2) l'interopérabilité de cette représentation avec la recommandation RIF du W3C, (3) leur publication sur le Web de données et (4) la réutilisation de ces règles basée sur l'interrogation de sources de données RDF représentant des règles à l'aide de requêtes SPARQL. Nous avons construit un ensemble de requêtes SPARQL qui permettent (1) la construction de bases de règles spécifiques à un contexte ou une application particuliers, (2) l'optimisation des raisonnements par la sélection des seules règles pertinentes pour un jeu de données, (3) la validation de bases de règles par rapport à une source de données RDF et (4) la mise à jour de bases de règles.

Plan

Le chapitre 2 présente les notions de base de la représentation des connaissances et du raisonnement, puis les modèles de représentation des connaissances sur le Web sémantique.

Le chapitre 3 présente notre étude bibliographique relative à notre travail de recherche. Cet état de l'art recense les langages de règles pour le Web sémantique, et les travaux sur la recommandation, l'échange et la réutilisation de règles sur le Web.

Le chapitre 4 présente notre positionnement par rapport au standard RIF, nous définissons la transformation d'une base de règles SPARQL dans un dialecte RIF ainsi que la transformation inverse [Seye 2012a], [Seye 2012b].

Le chapitre 5 présente notre approche de la réutilisation de règles pertinentes à partir de sources de règles existantes, en écrivant un jeu de requêtes SPARQL, pour sélectionner des règles par leur contenu ainsi que leurs méta-données [Seye 2014], construire automatiquement des bases de règles spécifiques à un contexte, un domaine. Nous aborderons également dans cette partie la validation de bases de règles par rapport à une ontologie ainsi que la mise à jour de bases de règles. Nous présentons le calcul de graphe de dépendances de règles.

Le chapitre 6 présente notre travail d'implémentation et d'évaluation des différentes méthodes proposées. Nous décrivons d'abord l'implémentation du dialecte RIF-SPARQL et les tests réalisés avec le moteur de règles de Corese/KGRAM. Nous présentons ensuite la mise en œuvre et l'évaluation de notre approche de publication et de réutilisation de règles en utilisant le moteur sémantique Corese/KGRAM qui permet d'interroger des sources de données distribuées et d'appliquer des règles d'inférence sur des sources de données distribuées. Finalement nous présentons notre implémentation de l'optimisation du moteur de règles basée sur le calcul de graphe de dépendance et nous montrons son impact sur le temps de calcul des inférences.

CHAPITRE 2

Préliminaires

2.1 Introduction

Ce chapitre est consacré aux notions de base de la représentation des connaissances et du raisonnement en général, puis sur le Web sémantique. La section 2 décrit deux approches de la représentation des connaissances : la première repose sur la représentation d'ontologies et la seconde sur la représentation de règles. La section 3 présente les modèles de représentation des connaissances du Web de données. La section 4 présente les modèles de raisonnement sur le Web sémantique.

2.2 Représentation des connaissances et raisonnement

La représentation de connaissances vise à formaliser des connaissances pour permettre aux agents logiciels d'effectuer des raisonnements afin de déduire de nouvelles connaissances et de répondre à des questions. Il existe deux approches principales de représentation de connaissances : d'une part, des modèles de représentation basés sur les ontologies, d'autre part, des représentations à base de règles.

2.2.1 Modèles à base d'ontologies

Le terme « Ontologie » est issu de la philosophie qui signifie l'étude de l'être. Il existe plusieurs définitions d'une ontologie en informatique. En Ingénierie des

Connaissances, une ontologie est utilisée pour définir les concepts qui permettent de décrire un domaine de connaissances ainsi que les relations entre ces concepts.

[Neches 1991] définit une ontologie comme suit : « *Une ontologie définit les termes et les relations de base comportant le vocabulaire d'un domaine ainsi que les règles afin de combiner les termes et les relations pour définir des extensions du vocabulaire* ».

La définition de [Gruber 1993] est la plus utilisée dans le domaine du Web sémantique : « *Une ontologie est une spécification explicite et formelle d'une conceptualisation d'un domaine de connaissance* ». La conceptualisation est la représentation abstraite du monde à représenter ; elle consiste en des relations entre concepts ainsi que des contraintes pour les employer.

Une ontologie permet de formaliser les connaissances générales d'un domaine. Elle capture la sémantique du domaine et des connaissances pour représenter les individus du domaine. Les ontologies permettent de définir les concepts d'un domaine, les classes et les propriétés, et de les organiser de manière hiérarchique. L'objectif principal d'une ontologie est le partage et la réutilisation de connaissances sur un domaine en établissant un consensus sur le vocabulaire conceptuel. Elle permet de faciliter le traitement des connaissances par des agents logiciels en fournissant une description partagée. Pour représenter et raisonner avec les connaissances d'une ontologie il existe deux grandes familles constituées des logiques de description et des graphes conceptuels.

2.2.1.1 Les logiques de description

Les logiques de description sont des langages pour la représentation des connaissances et du raisonnement. Elles constituent des sous-ensembles de la logique du premier ordre décidables (tous les problèmes peuvent être résolus en un temps fini). La représentation des connaissances en logiques de description distingue les connaissances de niveau terminologique, appelé TBox, et celles de ni-

veau factuel (instances), appelé ABox. La TBox est constituée du vocabulaire du domaine, des concepts (classes) et rôles (propriétés) et des relations de subsomption entre concepts et entre rôles. La ABox permet de décrire les individus.

La Figure 2.1 présente un extrait d'une ontologie visant à représenter une famille.

TBox :	**ABox :**
Animal $\equiv$ Femelle $\sqcup$ Male	Humain(Flora)
Humain $\sqsubseteq$ Animal	Femelle(Flora)
Femme $\equiv$ Humain $\sqcap$ Femelle	Femme(Mary)
Homme $\equiv$ Humain $\sqcap \neg$ Femme	Homme(John)
estParent(Humain, Humain)	estParent(Flora,John)
Mère $\equiv$ Femme $\sqcap \exists$ estParent	estParent(John,Mary)

FIGURE 2.1 – TBox et ABox de l'ontologie famille

Les logiques de description sont constituées de plusieurs familles réparties selon leur expressivité et complexité.

FL (Frame-based description Language) [Brachman 1984] et *EL* (Existential restrictions Language) [Baader 2005] sont les logiques de description les plus simples. *FL* et *EL* permettent d'exprimer la conjonction de concepts $C_1 \sqcap C_2$, les restrictions de concepts universelles $\forall R.C$ pour FL et les restrictions existentielles $\exists R.C$ pour EL.

La famille *ALC* (Attribute Language for Complex Concept) est une extension de la famille *EL* avec la négation. Elle permet ainsi en exploitant la négation d'exprimer l'union de concepts $(C_1 \sqcup C_2 \equiv \neg(\neg C_1 \sqcap \neg C_2))$ et les restrictions universelles $(\forall R.C \equiv \neg(\exists R.\neg C))$.

La famille *SHOIN* est une extension de *ALC* avec les restrictions de cardinalité, la définition de concepts à partir de constantes (énumération d'individus), la définition de rôles inverses, la transitivité et l'inclusion de rôles (rôle correspond à relation en logique de description).

Les logiques de description permettent de faire des raisonnements sur la TBox. Les raisonnements sur la TBox consistent à calculer les relations de subsomption

qui existent entre concepts ou entre rôles (classification), et à vérifier la consistance des connaissances représentées (absence de contradiction). Les logiques de description permettent de faire des raisonnements et des inférences à partir de la TBox et de la ABox, il s'agit de faire des raisonnements à partir d'une base de connaissances composée d'une TBox avec une ABox telles que la vérification d'une instance de concept, répondre à des requêtes.

2.2.1.2 Les graphes conceptuels

Le modèle des graphes conceptuels [Sowa 1984] permet de représenter les connaissances sous forme de réseau sémantique, muni d'une sémantique formelle. Comme les logiques de description, le modèle des graphes conceptuels permet de représenter, en les distinguant, les connaissances ontologiques d'un domaine et des connaissances factuelles basées sur cette ontologie. Le support définit le vocabulaire sur lequel seront basés les graphes conceptuels. Il permet de déclarer des types de concepts et des types de relations, des relations de subsomption entre types, et des axiomes. Les connaissances factuelles sont représentées par des graphes basés sur le support. Les graphes conceptuels comportent des nœuds concepts et des arcs étiquetés par des types de relation. Un concept est constitué d'un type de concept et d'un marqueur, individuel (qui désigne un individu) ou générique (qui correspond à une variable). Par exemple le nœud `[Animal:*]` représente un animal et `[Animal:bob]` représente l'animal bob. Le graphe conceptuel de la Figure 2.2 représente l'énoncé « Lucy va à la kermesse en portant une robe violette ».

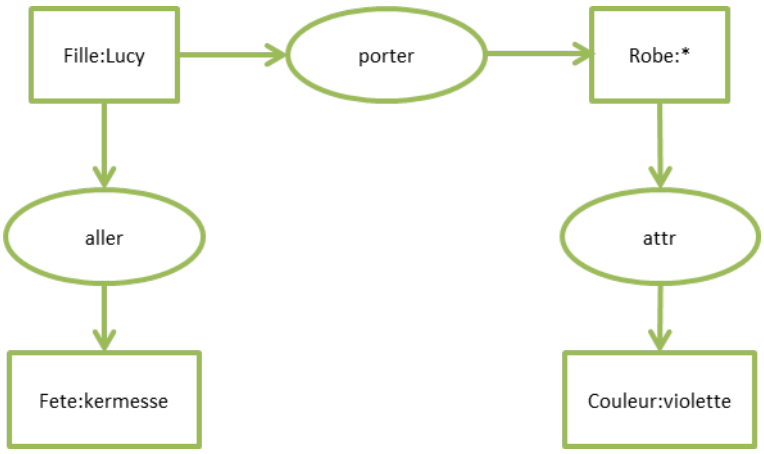

FIGURE 2.2 – Exemple de graphe conceptuel

Les graphes conceptuels sont basés sur la logique. Le raisonnement dans le modèle des graphes conceptuels peut être un raisonnement logique basé sur la traduction des graphes en logique du premier ordre. Dans ce cas les graphes conceptuels sont une représentation graphique de la logique du premier ordre. Les graphes conceptuels peuvent également être considérés comme une représentation à part entière [Mugnier 1996]. Dans ce cas les raisonnements sont basés sur un ensemble d'opérations de graphes, issues de la théorie des graphes, telles que la projection de graphe, la spécialisation et la restriction de concept ou de relation.

2.2.2 Modèles à base de règles

Les modèles de représentation des connaissances à base de règles associent des prémisses (conditions ou antécédents de la règle) à des conclusions (conséquent de la règle). Les conditions peuvent représenter soit des faits soit des situations, les conclusions représentent de nouveaux faits à déduire ou des actions à exécuter. Dans cette section nous présentons d'abord les différentes catégories de règles qui existent, puis nous nous centrons sur la programmation logique avec Datalog, et nous exposons les méthodes de raisonnements à base de règles.

2.2.2.1 Les catégories de règles

On distingue trois classes principales de règles : les règles déductives, les règles d'intégrité et les règles réactives.

Règles déductives Les règles déductives sont composées d'une ou de plusieurs conditions (les propositions d'entrée) et une conclusion (la proposition de sortie). Les règles déductives sont utilisées pour calculer des conséquences logiques par leur application sur des faits. En raisonnement déductif, la vérité des prémisses d'une règle implique logiquement la vérité de la conclusion. La déduction logique est basée sur la preuve formelle. A partir d'axiomes et d'hypothèses, de nouveaux faits sont déduits en appliquant des règles. Les règles déductives sont de la forme :
$\forall t(h(t) \longleftarrow \exists y(b_1(t,y) \wedge \ldots \wedge b_n(t,y)))$,
où $h(t)$ est le conséquent de la règle et $\exists y(b_1(t,y) \wedge \ldots \wedge b_n(t,y))$ l'antécédent de la règle. Par exemple, en considérant les deux énoncés

```
S'il est docteur ALORS  il a fait de longues études.
John est docteur.
```

on peut en déduire l'énoncé

```
John a fait de longues études.
```

Règles d'intégrité Des règles d'intégrité permettent de définir des contraintes sur les données. Elles ont pour rôle d'assurer la consistance d'une base de connaissances. Les règles d'intégrité permettent ainsi de vérifier que tous les énoncés d'une base de connaissances sont corrects.

Voici un exemple de règle d'intégrité :

```
Un doctorant ne peut être ATER que s'il est en dernière année de doctorat.
```

et une base de connaissances qui viole la règle énoncée ci-dessus :

```
John est ATER.
John est en première année de doctorat.
```

Règles réactives Les règles réactives sont utilisées dans les programmes à base de règles de systèmes réactifs. Les systèmes réactifs permettent de détecter automatiquement un événement et d'y répondre en un temps limité. Il existe deux types de règles réactives : des règles événement-condition-action et des règles de production.

Les règles événement-condition-action (ECA) spécifient d'exécuter automatiquement une action si un événement est détecté et si la condition est vérifiée. Elles sont de la forme :

```
LORSQUE un événement E se produit,
SI la condition C est vraie
ALORS faire l'action A.
```

Exemple de règle pour les opérations sur un terminal bancaire :

```
LORSQUE la touche validation est tapée
SI code correct
ALORS  afficher menu (opérations possibles)
```

Les règles de production spécifient de faire une action si une mise à jour de la base de connaissances conduit à la vérité de la condition. Elles sont de la forme :

```
SI condition ALORS FAIRE action.
```

Exemple :

```
SI un client dépense plus de 100 euro
ALORS FAIRE  une réduction de 15 euro
```

2.2.2.2 Datalog

Datalog est un langage logique permettant de définir des règles logiques et des requêtes pour les bases de données déductives. Il sert à manipuler logiquement des données et les relations entre ces données. Il est défini comme le langage logique le plus simple pour le modèle relationnel. Un programme Datalog est constitué

d'un ensemble de faits et règles. Ces règles permettent de définir les relations entre les faits et aussi de trouver de nouvelles informations à partir de celles existantes. Datalog permet d'exprimer tous les opérateurs relationnels (restriction, jointure, projection, ...) et la récursivité.

Datalog a une syntaxe proche des clauses de Horn de Prolog. Les termes de Datalog sont les variables, les constantes et les prédicats. Les termes de Datalog sont construits à partir de constantes individuelles, de constantes symboles de fonction et de variables. Datalog admet deux types de connecteurs, l'implication $\longleftarrow$ et la conjonction $\wedge$. Une règle Datalog est de la forme H $\longleftarrow$ B, avec H un littéral positif constituant la tête de la règle (ou conclusion) et B une conjonction de littéraux positifs ($L_1 \wedge \ldots \wedge L_n$) constituant le corps de la règle (ou prémisse), tel que toutes les variables de H apparaissent dans B. Une règle sans antécédent est un fait. La règle « *Si X est frère de Y et Y père de Z, alors X est oncle de Z* » est représentée en Datalog comme suit :

`oncle(X,Z)` $\longleftarrow$ `frere(X,Y)` $\wedge$ `pere(Y,Z).`

Voici un exemple de règle récursive :

`ancêtre(X,Y)` $\longleftarrow$ `ancêtre(X,Z)` $\wedge$ `parent(Z,Y)`

Datalog n'admet pas de quantificateur dans la définition des règles ni de négation. Toutefois, il existe des extensions pour supporter la négation et les objets.

2.2.2.3 Déduction logique

La déduction logique sur un ensemble de règles et de faits peut s'exprimer dans les termes de la théorie des preuves ou de la théorie des modèles. Dans la théorie des preuves, la déduction est syntaxique : une formule logique est une conséquence *syntaxique* d'un ensemble de formules. Dans la théorie des modèles, une sémantique est associée aux formules logiques, par le biais d'une interprétation et une formule logique est une conséquence *sémantique* d'un ensemble de formules. Les programmes logiques reposent sur la théorie des preuves.

Lla recherche de la solution d'un programme P à partir des faits de P repose sur la déduction de nouveaux faits en appliquant des règles d'inférence : l'instanciation de règles universelles, le « modus ponens » et l'évaluation de formules conjonctives ou disjonctives.

Instanciation Considérons par exemple la règle universelle représentant la proposition bien connue que "tout homme est mortel" :

```
Forall ?x (bio:mortal(?x) :- bio:human(?x))
```

Cette règle universelle peut être instanciée en substituant à la variable `?x` n'importe quelle donnée représentant un individu. Par exemple, en remplaçant `?x` par `phil:Socrates` représentant un célèbre individu, on en déduit la règle :

```
bio:mortal(phil:Socrates) :- bio:human(phil:Socrates)
```

Modus ponens Considérons maintenant une base de connaissances comportant la règle universelle ci-dessus et le fait représentant que "Socrate est un homme" :

```
Forall ?x (bio:mortal(?x) :- bio:human(?x)
bio:human(phil:Socrates) )
```

La prémisse de la règle universelle peut être instanciée en remplaçant la variable `?x` par la constante `phil:Socrates` et ainsi être vérifiée par le fait du document. La règle du modus ponens permet alors de déduire un nouveau fait construit par instanciation de la conclusion de la règle en remplaçant la variable `?x` par la même constante `phil:Socrates`. On obtient ainsi le fait `bio:mortal(phil:Socrates)`.

Evaluation des formules conjonctives et disjonctives Dans le cas où des faits ou des prémisses de règles ne sont plus des formules atomiques comme dans les exemples ci-dessus, des règles d'évaluation permettent d'évaluer leur valeur de

vérité : une formule conjonctive est vraie si toutes les formules en conjonction sont vraies ; une formule disjonctive est vraie si l'une au moins des formules en disjonction est vraie.

Ces trois briques de raisonnement permettent d'opérationaliser la sémantique des règles d'inférence par une méthode de déduction dite de *chaînage avant* ou par une méthode dite de *chaînage arrière*. Le raisonnement par chaînage avant consiste à traiter récursivement les règles en partant de leurs prémisses pour en conclure de nouveaux faits qui viennent enrichir la base de faits. Pour ce faire on peut appliquer la méthode suivante jusqu'à ce qu'il n'y ait plus de fait déductible :

- Partir des faits
- Évaluer les formules conjonctives
- Appliquer la règle d'instanciation (déduire tous les faits possibles)
- Appliquer la règle du modus ponens (déduire tous les faits possibles)

Ensuite le fait à prouver est recherché dans les faits déduits.

Le raisonnement par chaînage arrière consiste à prouver un fait en considérant les règles en partant de leurs conclusions pour remonter aux faits contenus dans la base.

- Partir de la conclusion
- Sélectionner les règles qui peuvent la prouver
- Vérifier les prémisses de ces règles récursivement

On remonte ainsi jusqu'aux faits stockés dans la base. Si des faits sont trouvés dans la base, le fait considéré est prouvé. C'est une méthode guidée par le but.

2.3 Web de données

Le Web de données, imaginé par Tim Berners-Lee, permet de publier sur le Web des connaissances dans un format compréhensible par des agents logiciels. Il est basé sur un ensemble de standards recommandés par le W3C : URI pour nommer les ressources, RDF pour les décrire et SPARQL pour interroger les données RDF.

2.3.1 URI

Uniform Resource Identifier [1] (URI) est un modèle commun proposé par Tim Berners-Lee et al. pour identifier de façon unique les ressources du Web. Il permet de nommer une ressource ou de désigner sa localisation sur le Web de données. Par exemple, `http://sws.geonames.org/2993457/` identifie Monaco dans Geonames (contenant des données géographiques). Pour décrire des données sur le Web, il est d'abord nécessaire de les identifier. L'URI constitue ainsi l'élément de base du Web en général et du Web de données en particulier en permettant d'exprimer tous les hyperliens du Web.

2.3.2 RDF

Resource Description Framework (RDF) est le modèle de base pour la représentation de connaissances sur le Web de données. RDF est destiné à décrire de façon formelle les ressources du Web et leurs métadonnées pour permettre leur traitement automatique. RDF favorise l'interopérabilité. En effet, il permet à une communauté d'utilisateurs de partager les mêmes métadonnées pour des ressources partagées. En RDF, les ressources sont associées à des URI pour les identifier. Il existe plusieurs syntaxes pour RDF dont RDF/XML, la première syntaxe recommandée, et Turtle dont la lecture est plus facile pour les utilisateurs humains. RDF permet de définir des relations entre deux ressources ou entre une ressource et un littéral. RDF représente les données sous forme de graphe. Un graphe RDF est un ensemble de triplets ; chaque triplet RDF est composé de trois éléments : un sujet, un prédicat et un objet. Le sujet d'un triplet est une ressource identifiée par un URI ou une ressource anonyme (ou blank node), qui joue le rôle de variable existentielle. Le prédicat est la propriété (une ressource) qui relie le sujet à l'objet. L'objet est une ressource (identifiée par un URI ou anonyme) ou une valeur littérale.

La Figure 2.3 représente un graphe RDF. La Figure 2.4 est une représentation

1. `http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/`

de ce graphe dans la syntaxe Turtle et la Figure 2.5 en est une représentation en RDF/XML.

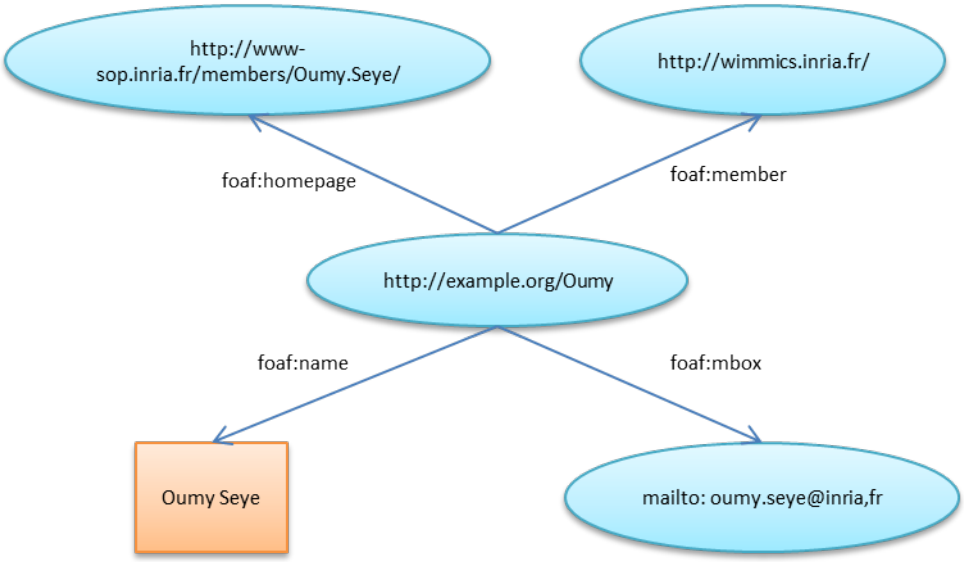

FIGURE 2.3 – Exemple de graphe RDF

```
@prefix ex: <http://example.org/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
ex:Oumy foaf:homepage <http://www-sop.inria.fr/members/Oumy.Seye/>;
   foaf:name "Oumy Seye";
   foaf:member <http://wimmics.inria.fr>;
   foaf:mbox <mailto:oumy.seye@inria.fr>.
```

FIGURE 2.4 – Graphe RDF au format Turtle

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <rdf:Description rdf:about="http://example.org/Oumy">
     <foaf:homepage rdf:resource="http://www-sop.inria.fr/members/
        Oumy.Seye/"/>
     <foaf:name>Oumy Seye</foaf:name>
     <foaf:member rdf:resource="http://wimmics.inria.fr"/>
     <foaf:mbox rdf:resource="mailto:oumy.seye@inria.fr"/>
  </rdf:Description>
</rdf:RDF>
```

FIGURE 2.5 – Graphe RDF au format RDF/XML

2.3.3 SPARQL

SPARQL est le langage recommandé par le W3C [2] pour interroger des graphes RDF. SPARQL 1.0 [3] est la première recommandation pour interroger des données RDF. Elle définit des requêtes de le forme SELECT, ASK, DESCRIBE et CONSTRUCT. Une requête de la forme SELECT renvoie la liste des associations des variables de la requête à des ressources ou littéraux pour lesquelles il existe un appariement du graphe de la clause WHERE de la requête avec le graphe RDF interrogé. Une requête SPARQL de la forme ASK permet de poser des requêtes à réponse booléenne sur un graphe RDF ; elle permet de demander si un appariement existe entre le graphe requête et le graphe RDF ; elle commande la recherche d'un appariement entre le graphe de sa clause WHERE et le graphe RDF interrogé. Une requête SPARQL de la forme CONSTRUCT produit un nouveau graphe RDF en remplaçant les variables du graphe de la clause CONSTRUCT par les valeurs pour lesquelles le graphe requête de la clause WHERE s'apparie avec le graphe RDF interrogé. Une requête SPARQL de la forme DESCRIBE permet de demander la description RDF d'une ressource présente dans les données RDF stockées ; son résultat est un ensemble de triplets RDF décrivant la ressource ciblée.

2. http://www.w3.org/TR/sparql11-overview/
3. http://www.w3.org/TR/rdf-sparql-query/

La clause WHERE des requêtes SPARQL de la forme SELECT, ASK ou CONSTRUCT contient un patron de graphe construit comme une conjonction de triplets où une variable peut prendre la place d'une ressource ou d'un littéral. Ce peut également être une union de graphes, certaines parties du graphe requête peuvent être déclarées optionnelles, des filtres peuvent être appliqués aux solutions d'un appariement :

- `A UNION B` permet de réunir les solutions des graphes `A` et `B` ;
- `A OPTIONAL B` permet de retourner les solutions de `A`, qu'il y ait une solution avec `B` ou non ;
- `A FILTER B` permet de restreindre les solutions de `A` au moyen d'une expression booléenne `B`.

Des opérateurs permettent de modifier la séquence des solutions :

- DISTINCT permet d'éliminer les doublons ;
- ORDER BY permet de trier les solutions ;
- LIMIT permet de couper le nombre de solutions en limitant le nombre de solutions retournées ;
- OFFSET permet de couper le nombre de solutions en indiquant de commencer les solutions générées après le nombre de solutions indiqué.

Depuis le 21 Mars 2013 SPARQL 1.1 [4] est la recommandation pour interroger des données RDF. Elle apporte beaucoup de nouveautés à la version précédente de SPARQL, notamment :

- L'agrégation : différentes fonctions d'agrégation, COUNT, SUM, MIN, MAX, AVG, SAMPLE, GROUP_CONCAT, GROUP BY, HAVING, permettent d'effectuer des calculs sur un ensemble de résultats ;
- La projection d'expression : elle permet de faire l'assignation et la création de nouvelles valeurs avec le constructeur `AS`. Elle peut être utilisée avec des agrégats ;

  Exemple de requête utilisant l'agrégat AVG et le constructeur AS :

4. http://www.w3.org/TR/sparql11-overview/

```
PREFIX ex: <http://www.example.org/humans#>
SELECT (AVG(?age) AS ?agemoyen)
WHERE {
  ?x ex:age ?age
}
```

– La négation : `NOT EXISTS` et `MINUS` permettent d'exprimer la négation. `NOT EXISTS` permet de tester l'absence d'un certain pattern de graphe dans les graphes solutions recherchés et `MINUS` permet de supprimer de l'ensemble des graphes solutions ceux qui suivraient un certain pattern.

Exemple d'utilisation de `NOT EXISTS` :

```
PREFIX ex: <http://www.example.org/humans#>
SELECT ?x
WHERE {
  ?x a ex:Person
  FILTER NOT EXISTS { ?x ex:hasChild ?y }
}
```

Exemple d'utilisation de `MINUS` :

```
PREFIX ex: <http://www.example.org/humans#>
SELECT ?x
WHERE {
  ?x a ex:Person
  MINUS { ?x ex:hasChild ?y }
}
```

– Les chemins de propriétés : un chemin définit une "voie" possible entre deux nœuds d'un graphe ;

Exemple d'un chemin impliquant une même propriété un nombre quelconque de fois :

```
PREFIX ex: <http://www.example.org/humans#>
SELECT ?y
WHERE { ?x ex:hasParent+ ?y }
```

– Les sous-requêtes : une sous-requête est une requête de la forme SELECT imbriquée dans une autre et dont les résultats sont utilisés dans la requête principale.

  Exemple de requête imbriqué permettant de rechercher toutes les ressources dans le namespace de OWL pour ensuite rechercher dans la requête principale les requêtes (représentées en SPIN) dont la clause WHERE contient une telle ressource :

```
PREFIX sp: <http://spinrdf.org/sp#>
SELECT ?resource  ?a
WHERE {
  { SELECT DISTINCT ?resource
    WHERE {
      ?resource a rdfs:Resource
      FILTER (strstarts(?resource, owl:))
    }
  }
  ?a a sp:Construct
  ?a sp:where ?m
  ?m (!sp:nil)+ ?x
  ?x ?p ?resource
}
```

Aux côtés du langage de requête, SPARQL 1.1 Update permet de modifier des données RDF avec les opérateurs suivants :

– `CREATE` et `DROP` pour créer ou supprimer des graphes ;
– `INSERT DATA` pour ajouter des données sans variables ;
– `DELETE DATA` pour supprimer des données sans variables ;
– `INSERT` pour ajouter des triplets dans un graphe ;
– `DELETE` pour supprimer des triplets dans un graphe .

2.3.4 Web de données liées

Le Web de données liées « Linked Open data » [Bizer 2009] a été initié par le W3C. C'est un principe qui favorise la publication de données structurées sur le Web en les reliant entre elles de façon typée pour constituer un réseau global d'informations pour le partage des données sur le Web. Il a pour objectif d'augmenter la visibilité et la réutilisation de données. Le Web de données liées repose sur quatre règles définies par Tim Berners-Lee :

- Utiliser des URI pour identifier les individus à décrire sur le Web.
- Utiliser des URI HTTP (déréférençables) pour permettre l'accès aux données décrivant les ressources ainsi identifiées par des requêtes HTTP.
- Lorsqu'un URI est déréférencé, renvoyer des données respectant les standards, de préférence en RDF.
- Lier les URI entre elles, lier les sources de données publiées aux sources existantes, pour construire ainsi un Web de données liées dans lequel on puisse découvrir des informations en navigant à partir d'un URI d'entrée.

2.4 Web sémantique

Pour raisonner avec les connaissances publiées sur le Web de données, le Web Sémantique propose dans sa pile de standards trois langages pour définir des vocabulaires pour décrire les données RDF : RDFS, OWL et SKOS. En apportant de la sémantique aux données RDF, ils permettent de raisonner sur ces données. C'est en ce sens que le Web de données devient Web sémantique. Nous présentons dans cette partie RDFS qui permet de définir des ontologies dites légères, OWL qui permet de définir des ontologies dites lourdes, et SKOS qui permet de représenter des systèmes d'organisation des connaissances tels que les thesaurus et les classifications.

2.4.1 RDFS

RDF Schema [5] (RDFS) fait partie de la recommandation RDF pour la description de son vocabulaire. RDFS permet de déclarer, en RDF, les propriétés et les classes utilisées pour décrire des données RDF et les relations hiérarchiques qui existent entre propriétés et entre classes. Plus précisément, RDFS permet de définir :

- une ressource comme étant une classe de type `rdfs:Class`. Une classe est une ressource représentant un ensemble de ressources dont la valeur de la propriété `rdf:type` est cette classe ;
- une ressource comme étant une propriété de type `rdf:Property` ;
- des hiérarchies de classes avec la propriété `rdfs:subClassOf` ;
- des hiérarchies de propriétés avec la propriété `rdfs:subPropertyOf` ;
- le domaine et la portée d'une propriété avec les propriétés `rdfs:domain` et `rdfs:range` ;
- les labels des classes et propriétés avec la propriété `rdfs:label` ;
- les définitions en langue naturelle des classes et propriétés avec la propriété `rdfs:comment`.

La Figure 2.6 définit la propriété `foaf:homepage` du graphe de la Figure 2.3, cela signifie que le sujet de cette propriété sera typé par `foaf:Person` et que sa valeur sera typée par `foaf:Document`. Par exemple la ressource `<http://www-sop.inria.fr/members/Oumy.Seye/>` est de type `foaf:Document` et la ressource `<http://example.org/Oumy>` est de type `foaf:Person`.

5. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/

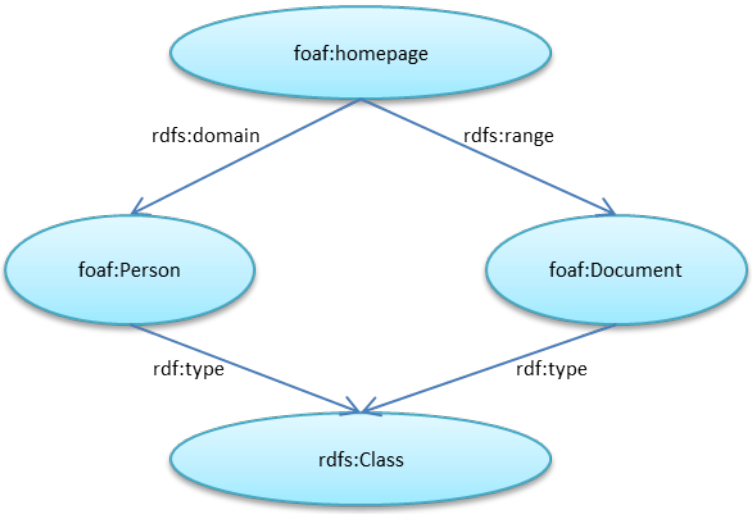

FIGURE 2.6 – Exemple de schéma RDFS

La sémantique de RDFS est définie par Patrick Hayes et ses coauteurs [Hayes 2014b] par un ensemble de triplets axiomatiques et de règles d'inférences. Ces règles d'inférence sont à la base du *régime d'inférence RDFS*[6] (en anglais, RDFS entailment). L'application de ces règles en chaînage avant sur un graphe de données RDF permet d'enrichir ce graphe par des triplets inférés. En chaînage avant ou arrière, elles permettent de vérifier si un graphe RDF implique logiquement un autre graphe RDF.

L'implémentation du régime d'inférence RDFS par un moteur sémantique lors de l'interrogation de données RDF permet de retrouver des réponses à des requêtes sans qu'elles soient explicites dans les données. Par exemple, il doit être déduit le fait que Mary est un humain à partir du fait que Mamy est une étudiante et de la connaissance ontologique que la classe des étudiants est une sous classe de celle des humains :

```
ex:Mary rdf:type ex:Etudiant
ex:Etudiant rdfs:subClassOf ex:Humain
```

6. http://www.w3.org/TR/sparql11-entailment/#RDFSEntRegime

Ainsi, grâce à l'inférence du triplet `ex:Mary rdf:type ex:Etudiant`, `Mary` sera retrouvée comme solution à la requête SPARQL suivante :

```
SELECT ?humain
WHERE { ?humain a ex:Humain }
```

2.4.2 OWL

Pour définir des ontologies riches, l'expressivité de RDFS devient insuffisante. En effet RDFS ne permet pas d'exprimer l'unicité, l'union, l'intersection de classes, ni certaines propriétés algébriques de propriétés telles que la transitivité, la symétrie, etc. C'est ce qui a motivé la recommandation de Web Ontology Language (OWL) qui permet non seulement de déclarer des propriétés et des classes mais aussi de les définir.

OWL est muni de plusieurs syntaxes :

– une syntaxe fonctionnelle [7],

  Exemple :

```
equivalentClass(:Doctorant :Thesard)
```

– une syntaxe XML [8],

  Exemple :

```
<EquivalentProperty>
  <ObjectProperty IRI="#Maker"/>
  <ObjectProperty IRI="#Creator"/>
</EquivalentProperty>
```

– la syntaxe de Manchester [9] basée sur les frames,

  Exemple :

```
ObjectProperty: Maker
  EquivalentProperty: Creator
```

7. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html#2.1
8. http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/
9. http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/

– une syntaxe RDF [10], pour partager des ontologies OWL sur le Web.

Exemple :

```
<owl:ObjectProperty rdf:ID='Maker'>
  <owl:EquivalentProperty rdf:resource='#Creator' />
</owl:ObjectProperty>
```

Voici la liste des principaux énoncés que permet d'exprimer OWL :

– Equivalence de classes, en utilisant la propriété `owl:equivalentClass`,

```
<owl:Class rdf:ID="Doctorant">
  <owl:equivalentClass rdf:resource="#Thesard"/>
</owl:Class>
```

– Equivalence de propriétés, en utilisant la propriété `owl:equivalentProperty`,

```
<owl:ObjectProperty rdf:about="Maker">
  <owl:equivalentProperty rdf:resource="#Creator"/>
</owl:ObjectProperty>
```

– Relations inverses, en utilisant la propriété `owl:inverseOf`,

```
<rdf:Property rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</rdf:Property>
```

– Relations symétriques, en utilisant la classe `owl:SymmetricProperty`,

```
<owl:SymmetricProperty rdf:about="#hasFriend"/>
```

– Relations transitives, en utilisant la classe `owl:TransitiveProperty`,

```
<owl:TransitiveProperty rdf:ID="hasAncestor"/>
```

– Restrictions de propriétés, en utilisant la classe `owl:Restriction` et la propriété `owl:onProperty`,

  – Contraintes de valeurs, en utilisant les propriétés `owl:allValuesFrom`, `owl:someValuesFrom`, `owl:hasValue`

10. http://www.w3.org/TR/owl2-mapping-to-rdf/

```
<owl:Restriction>
  <owl:onProperty rdf:ID="hasParent"/>
  <owl:allValuesFrom rdf:resource="#Human"/>
</owl:Restriction>
```

– Contraintes de cardinalité, en utilisant les propriétés `owl:maxCardinality` et `owl:minCardinality`,

```
<owl:Restriction>
  <owl:onProperty rdf:ID="hasParent"/>
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
     2</owl:minCardinality>
</owl:Restriction>
```

– Union de classes, en utilisant la propriété `owl:unionOf`,

```
<owl:Class rdf:ID="Academic">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Researcher"/>
    <owl:Class rdf:about="#Lecturer"/>
  </owl:unionOf>
</owl:Class>
```

– Intersection de classes, en utilisant la propriété `owl:intersectionOf`.

```
<owl:Class rdf:ID="Professor">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Lecturer"/>
    <owl:Class rdf:about="#Researcher"/>
  </owl:intersectionOf>
</owl:Class>
```

OWL est issu de la communauté des logiques de descriptions et est conçu comme une famille de langages. La première recommandation de OWL, OWL 1 [11], comporte trois sous-langages d'expressivités différentes :

– OWL Lite est le sous-langage le plus simple de OWL. Il permet d'exprimer des hiérarchies de classes et de propriétés, certaines propriétés algébriques

11. http ://www.w3.org/TR/owl-features/

de propriétés et des contraintes de cardinalité 0 et 1 sur les propriétés.

– OWL DL est plus expressif que OWL Lite. Il inclut toutes les propriétés de OWL. Toutefois, un ensemble de contraintes a été fixé dans l'utilisation de certaines afin de garantir la complétude (tous les raisonnements terminent) et la décidabilité (tous les raisonnements terminent en temps fini). Notamment il est nécessaire de faire une séparation entre classes, propriétés et instances.
– OWL Full offre une expressivité maximum. Il est compatible avec RDFS alors que les deux premières familles définies ci-dessus ne le sont pas puisque RDFS ne distingue pas individus, classes et propriétés. Toutefois, OWL Full ne garantit pas la complétude et la décidabilité.

La recommandation OWL 2 [12] publiée le 11 Décembre 2012 comporte trois sous-langages ou *profils* :

– OWL 2 EL (OWL Existential restrictions Language) est un profil de OWL 2 conçu avec une expressivité réduite pour raisonner (classifier) efficacement sur de grosses ontologies, comportant beaucoup de classes et propriétés. OWL 2 EL est basé sur la famille EL des logiques de description.
– OWL 2 QL (OWL Query Language) est un profil de OWL 2 conçu pour interroger efficacement (en LOGSPACE) de gros graphes de données basés sur des ontologies relativement légères, avec des requêtes conjonctives et en utilisant les technologies standards des bases de données relationnelles. OWL 2 QL est basé sur la famille DL-Lite des logiques de description [Calvanese 2007].
– OWL 2 RL (OWL Rule Language) est un profil de OWL 2 qui peut être implémenté par un calcul en temps polynomial par un programme logique (qui manipule un ensemble de règles d'inférence, d'où l'acronyme RL pour Rule Language).

Les raisonnements classiques issus des logiques de description sont la classification (calculer les relations de subsomption entre classes ou entre propriétés),

12. http ://www.w3.org/TR/owl2-overview/

l'identification (déterminer de quelle(s) classe(s) un individu est instance) et la vérification de la cohérence de bases de faits relativement à une ontologie. Par exemple si deux classes sont déclarées disjointes, un individu ne peut être instance de ces deux classes. Ainsi les assertions suivantes sont inconsistantes.

```
<Male> owl:disjointWith <Femelle>.
<papisco> rdf:type <Male>.
<papisco> rdf:type <Femelle>.
```

2.4.3 SKOS

SKOS [13] (Simple Knowledge Organisation System) est une recommandation du W3C pour représenter et échanger des systèmes d'organisation de connaissances tels que des thésaurus, des classifications et des glossaires. SKOS permet ainsi de modéliser des vocabulaires qui n'ont pas atteint le statut d'ontologie ou bien il est utilisé en complément de RDFS ou OWL pour représenter des caractéristiques de thésaurus : des labels, leurs alignements, etc.

Voici les principaux constructeurs de SKOS :

– la classe `skos:Concept` permet de définir des concepts SKOS. `skos:Concept` est définie comme une instance de `owl:Class` ;

  Exemple :

  ```
  ex:Education_Universelle rdf:type skos:Concept.
  ```

– les propriétés `skos:prefLabel`, `skos:altLabel` et `skos:hiddenLabel` permettent d'assigner à un concept SKOS différents labels lexicaux, respectivement préférés, alternatifs et cachés. La valeur de la propriété `skos:prefLabel` d'un concept SKOS doit être unique pour chaque concept SKOS pour une langue donnée (un concept peut avoir différentes valeurs de propriété `skos:prefLabel` pour des langues différentes). La propriété `skos:hiddenLabel` permet d'associer à un concept SKOS des mauvaises

13. http://www.w3.org/TR/2009/REC-skos-reference-20090818/

orthographes connues.

Exemple :

```
ex:Education_Universelle rdf:type skos:Concept;
  skos:prefLabel "Education Universelle"@fr,
                 "Universal Education"@eng;
  skos:altLabel "Education Pour Tous"@fr;
  skos:hiddenLabel "Education Universel"@fr.
```

– les propriétés `skos:broader` et `skos:narrower` permettent d'exprimer des liens hiérarchiques entre deux concepts SKOS.

Exemple :

```
ex:Education_Universelle skos:broader ex:Droit_Education;
  skos:narrower ex:Scolarite_Obligatoire.
```

– la propriété `skos:related` est utilisée pour définir un lien associatif entre deux concepts ;

Exemple :

```
ex:Education_Universelle skos:related ex:Education_Integratrice.
```

– la propriété `skos:definition` permet d'associer à un concept SKOS une définition en langue naturelle.

Exemple :

```
ex:Education_Universelle skos:definition  "Système d'éducation
   rendant l'enseignement accessible à tous"@fr.
```

2.5 Conclusion

Nous avons présenté dans ce chapitre les grandes familles de modèles de représentation de connaissances et ceux du Web sémantique qui s'inscrivent dans la lignée des modèles de représentation à base de graphe et des logiques de description. Au regard à la fois des modèles de représentation de connaissances existants et de la pile des standards du Web sémantique définie par le W3C, il manque dans

cet exposé préliminaire, les modèles de représentation à base de règles pour le Web sémantique. C'est ce dont nous traitons dans la suite de cette thèse.

CHAPITRE 3

État de l'art

3.1 Introduction

Les règles constituent une alternative ou un complément pour les ontologies RDFS et OWL. Elles permettent de définir certaines contraintes qui ne peuvent pas être exprimées dans ces langages. La recherche dans le domaine du Web sémantique s'intéresse de plus en plus aux langages de règles, et plusieurs langages de règles ont vu le jour tels que RuleML [Boley 2001], SWRL [Horrocks 2004], Flora-2 [Yang 2003], Triple [Sintek 2002], N3Logic [Berners-Lee 2008a], ERML [Lee 2003] ...Ces langages ont des syntaxes et sémantiques différentes, ce qui pose des problèmes d'interopérabilité et d'échange qui sont des éléments clés du Web sémantique. Le langage RIF répond à ce problème ; c'est la recommandation du W3C pour l'échange de règles sur le Web, permettant l'interopérabilité des moteurs de règles existants.

Ce chapitre présente une vue d'ensemble des langages de règles pour le Web sémantique. La section 2 présente les langages de règles utilisés sur le Web sémantiques. La section 3 présente la recommandation RIF d'échange de règles pour le Web sémantique. la section 4 aborde les approches de la réutilisation de règles sur le Web. La section 5 introduit notre positionnement par rapport à l'état de l'art.

3.2 Langages de règles

3.2.1 Triple

Triple [Sintek 2002] est un langage de règles et de requêtes pour RDF, inspiré de F-Logic [Kifer 1995]. Triple constitue l'un des premiers langages de règles pour RDF basé sur les clauses de Horn. Triple permet d'effectuer des requêtes, des transformations et la dérivation de nouvelles connaissances à partir de données RDF. Ce langage de règles offre un langage unifié pour représenter les données RDF et les règles qui permettent de raisonner sur ces données. Triple s'inspire du langage F-Logic. Les triplets RDF sont représentés en Triple par des frames. Triple permet d'exprimer l'agrégation, la réification et aussi des expressions de chemin. Pour représenter des règles, Triple permet d'utiliser des connecteurs (négation, conjonction et disjonction) et les quantificateurs existentiel et universel.

```
FORALL X Y Z
  X[ex:hasUncle -> Z] <-
  X[ex:hasParent -> Y] AND Y[ex:hasBrother -> Z]
```

FIGURE 3.1 – Exemple de règle Triple

3.2.2 RuleML

RuleML [Boley 2001] est une famille de langages utilisée pour publier sur le Web différentes familles de règles telles que les règles de réaction, les règles de transformation, les règles de dérivation, ainsi que les faits, les requêtes et les contraintes d'intégrité. Chaque catégorie de règles de RuleML comporte une syntaxe particulière. La règle de la Figure 3.2 permet de représenter en ruleML que le frère de l'un des parents est un oncle. Un élément `if` contient la condition (Si `x1` a pour parent `x2` et `x2` a pour frère `x3`) et un élément `then` contient la conclusion (alors `x1` a pour oncle `x3`).

```
<Implies>
  <if>
    <And>
      <Atom>
        <Rel>hasParent</Rel>
        <var>x1</var>
        <var>x2</var>
      </Atom>
      <Atom >
        <Rel>hasBrother</Rel>
        <var>x2</var>
        <var>x3</var>
      </Atom>
    </And>
  </if>
  <then>
    <Atom>
      <Rel>hasUncle</Rel>
      <var>x1</var>
      <var>x3</var>
    </Atom>
  </then>
</Implies>
```

FIGURE 3.2 – Exemple de règle de déduction RuleML

3.2.3 SWRL

SWRL (Semantic Web Rule Language) [Horrocks 2004] est un langage de règles pour le Web sémantique basé sur la combinaison de OWL-DL et le sous-ensemble de RuleML basé sur les clauses de Horn (Datalog). SWRL est une extension de OWL-DL avec les règles de Horn sans fonction. Les règles SWRL ont la structure de règles Datalog « antécédant → conséquent » où l'antécédent et le conséquent de la règle peuvent consister en zéro ou plusieurs atomes. Toutefois, une règle avec plusieurs conséquents peut toujours être transformée en plusieurs règles avec un conséquent chacune. En SWRL, seuls des prédicats unaires et binaires peuvent être utilisés, i.e. tout atome est de la forme C(x) ou P(x,y).

SWRL a une syntaxe de présentation XML combinant les espaces de noms de `ruleml` et `swrlx`. SWRL a une librairie de fonctions prédéfinies lui permettant

de supporter une grande variété de fonctions et prédicats. La Figure 3.3 reprend l'exemple de règle utilisé pour illustrer RuleML (Figure 3.2) et montre comment l'exprimer en SWRL.

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

FIGURE 3.3 – Exemple de règle en SWRL

3.2.4 SILK

SILK (Semantic Inferencing for Large Knowledge) [Grosof 2009] propose un langage de règles pour le Web sémantique. SILK est construit à partir des règles de Horn et permet d'exprimer la conjonction et la négation dans la partie conséquent de la règle, la négation (neg) et la négation par l'échec (naf) dans la partie condition, l'exclusion mutuelle (!-), la réification, l'agrégation. Le langage SILK est muni de plusieurs types de connecteurs tels que l'implication gauche, l'implication droite et la double implication.

SILK introduit la notion de priorité entre prédicats, permettant, en cas de conflit, d'exécuter le prédicat prioritaire avant les autres.

3.2.5 SPARQL comme langage de règles

Une requête SPARQL de la forme CONSTRUCT produit un nouveau graphe RDF en remplaçant les variables du graphe de la clause CONSTRUCT par les valeurs pour lesquelles le graphe requête de la clause WHERE s'apparie avec le graphe RDF interrogé. Une telle requête peut être vue comme une règle et son traitement comme l'application d'une règle en chaînage avant pour enrichir le graphe RDF.

Exemple de règle SPARQL :

```
PREFIX  ex: <http://example.org/>
CONSTRUCT {?x1 ex:hasUncle ?x3}
WHERE {
  ?x1 ex:hasParent ?x2.
  ?x2 ex:hasBrother ?x3.
}
```

SPARQL est ainsi utilisé comme langage de règles dans plusieurs travaux sur le Web Sémantique. Dans [Polleres 2007a] et [Angles 2008], les auteurs on établi une correspondance entre SPARQL et Datalog (nr-Datalog$^{\neg}$, Datalog sans récursion et avec négation).

Dans [Schenk 2008], S. Chenk et al. utilisent SPARQL comme langage de règles pour définir de nouvelles données RDF à partir de sources de données existantes. La forme CONSTRUCT de SPARQL est utilisée dans des réseaux de graphes pour effectuer la correspondance de patrons de graphes sur des graphes d'entrée.

Le framework R2R [Bizer 2010], qui permet de publier des mappings sur le Web, est basé sur la forme CONSTRUCT de SPARQL pour exprimer des transformations de données. R2R permet de traduire des données du Web vers un schéma local.

Avec SPARQL++ [Polleres 2007b], les auteurs utilisent le langage SPARQL comme un langage de règles pour exprimer des alignements entre vocabu-

laires RDF et proposent pour cela certaines extensions de la forme de requêtes CONSTRUCT.

3.2.6 SPIN

SPIN [1] (acronyme de SPARQL Inferencing Notation), permet de représenter des requêtes SPARQL sous forme de graphe RDF. SPIN est une *Member Submission* au W3C [2] depuis 2011. SPIN propose une syntaxe RDF pour SPARQL [3] et une ontologie pour représenter en RDF une modélisation impliquant des contraintes et des règles [4].

Une requête SPARQL est représentée par une ressource de type `sp:Query` ou l'une de ses sous-classes `sp:Select`, `sp:Ask` ou `sp:Construct`. Les clauses d'une requête sont également réifiées : une requête est liée à sa clause WHERE par une propriété `sp:where` ; une requête de la forme CONSTRUCT est liée à sa clause CONSTRUCT par une propriété `sp:templates`. Les triplets des clauses SELECT et CONSTRUCT sont également réifiés : les valeurs des propriétés `sp:where` et `sp:templates` sont des listes de ressources, chacune étant le sujet d'une propriété `sp:subject`, `sp:predicate` et `sp:object` dont les valeurs sont les sujets, propriétés et valeurs des triplets de la clause. La Figure 3.4 présente un exemple de règle représentée par une requête SPARQL de la forme CONSTRUCT formalisée en SPIN.

1. http://spinrdf.org/
2. http://www.w3.org/Submission/spin-sparql/
3. http://spinrdf.org/sp.html
4. http://spinrdf.org/spin.html

```
@prefix sp: <http://spinrdf.org/sp#> .
@prefix c: <http://www.inria.fr/acacia/comma#> .
[ a sp:Construct ;
  sp:templates (
    [
      sp:subject _:sb0 ;
      sp:predicate c:hasUncle ;
      sp:object _:sb1
    ]
  );
  sp:where (
    [
      sp:subject _:sb0 ;
      sp:predicate c:hasParent ;
      sp:object _:sb2
    ]
    [
      sp:subject _:sb2 ;
      sp:predicate c:hasBrother ;
      sp:object _:sb1
    ]
  );
]
_:sb0 sp:varName "x".
_:sb1 sp:varName "z".
_:sb2 sp:varName "y".
```

FIGURE 3.4 – Exemple de règle SPIN

Des requêtes SPARQL ainsi sérialisées en RDF peuvent être facilement partagées et réutilisées sur le Web de données, comme n'importe quelles données.

Notamment, la propriété `spin:constraint` est utilisée pour associer à une classe une contrainte exprimée par une requête SPARQL de la forme ASK ou CONSTRUCT, qui doit être vérifiée pour toutes ses instances. En représentant une contrainte par une requête de la forme CONSTRUCT, la violation de contrainte, instance de la classe `spin:ConstraintViolation`, peut être expliquée ainsi que sa résolution dans la clause CONSTRUCT.

La propriété `spin:rule` est utilisée pour associer à une classe une règle d'inférence exprimée par une requête SPARQL de la forme CONSTRUCT ou une règle de mise à jour exprimée par une opération SPARQL UPDATE INSERT/DELETE. Ces règles devront être appliquées à toutes les instances de la classe à laquelle elles

sont associées. Par défaut l'ordre d'application des règles est indéfini. Toutefois, SPIN définit des propriétés permettant de spécifier l'ordre d'exécution des règles.

3.3 RIF, la recommandation du W3C pour l'échange de règles

RIF [5], acronyme de Rule Interchange Format, est la recommandation du W3C pour l'échange de règles sur le Web en général et sur le Web sémantique en particulier, que ce soient des règles logiques ou des règles de production. RIF a été conçu pour assurer l'interopérabilité et la portabilité de différents langages et systèmes à base de règles.

RIF se présente sous la forme d'un ensemble extensible de dialectes dont trois sont définis dans la recommandation : RIF-BLD [6], RIF-PRD [7] et RIF-Core [8]. BLD est l'acronyme de Basic Logic Dialect : RIF-BLD permet de représenter des programmes logiques, c'est-à-dire des suites de règles d'inférence, sur des faits positifs (donc sans négation). Il correspond à la logique de Horn, avec l'égalité. Syntaxiquement, il est étendu avec des frames, des URI désignant des concepts et les types de données de XML Schema.

PRD est l'acronyme de Production Rules Dialects : RIF-PRD permet de représenter des règles de production.

RIF-Core est un noyau de primitives commun à RIF-BLD et RIF-PRD, correspondant à la logique de Horn sans symbole de fonction, c'est-à-dire à Datalog, avec une sémantique classique en logique du premier ordre. Tout autre dialecte de RIF est une extension de RIF-Core avec possibilité de fonctionnalités supplémentaires suivant les besoins de l'utilisateur. RIF-FLD (RIF Framework for Logic Dialect) est le framework de base pour la création de nouveaux dialectes logiques de RIF

5. http://www.w3.org/TR/2010/NOTE-rif-overview-20100622/
6. http://www.w3.org/TR/rif-bld/
7. http://www.w3.org/TR/rif-prd/
8. http://www.w3.org/TR/rif-core/

avec des fonctionnalités non présentes dans RIF-BLD. Un dialecte est un langage de règles avec une syntaxe et une sémantique bien définies.

3.3.1 RIF-Core

RIF-Core [Boley 2010a] est le dialecte principal de RIF. RIF-Core constitue le noyau commun de tous les dialectes RIF. Il est basé sur les clauses de Horn. RIF-Core est destiné à gérer l'échange et l'implémentation de règles de Horn. Pour pouvoir manipuler des objets simples et des frames, RIF-Core effectue une extension de la logique de Horn. RIF-Core utilise les IRIs comme identificateurs. RIF-Core présente une syntaxe logique proche de Prolog sous la forme `H:- B`, H constitue la tête appelée aussi le conséquent de la règle et B le corps ou l'antécédent de la règle. RIF-Core supporte un ensemble riche de types de données et built-ins (bibliothèque de fonctions et prédicats) qui sont définis dans [Polleres 2010], de même que les types de données de XML Schema.

3.3.2 RIF-BLD

RIF-BLD [Boley 2010c] est une extension de RIF-Core, avec des fonctions et prédicats à arguments nommés. RIF-BLD intègre les notions d'égalité, de typage, la notion de sous-classe sur la partie conclusion de la règle. RIF-BLD est adapté aux systèmes basés sur les règles avec chaînage arrière. RIF-BLD est basé sur la programmation logique et la logique du premier ordre. RIF-BLD comporte trois syntaxes : la syntaxe de présentation, la syntaxe XML et la syntaxe RDF [Hawke 2013]. Sa compatibilité avec RDF et OWL est aussi définie dans la recommandation [de Bruijn 2010].

La Figure 3.5 présente un exemple d'utilisation de RIF-BLD.

```
Document(
Prefix(h <http://www.inria.fr/humans#>)
Group (
  ForAll  ?x ?z ?y (
    ?x[ h:hasUncle -> ?z] :-
    And (
      ?x[ h:hasParent -> ?y]
      ?y[ h:hasBrother -> ?z] ) )

  h:Flora[ h:hasParent -> h:John]
  h:John[ h:hasBrother -> h:Mark] ) )
```

FIGURE 3.5 – Exemple de document RIF-BLD

Comme cela est classique en logique mathématique, la recommandation du W3C définit le langage de RIF-BLD comme l'ensemble de ses formules bien formées à partir de termes construits sur un alphabet. L'alphabet de RIF-BLD est constitué d'un ensemble *Const* de noms de constantes, d'un ensemble *Var* de noms de variables qui commencent par le caractère ' ?', d'un ensemble *ArgNames* de noms d'arguments, de connecteurs logiques *And*, *Or* et :-, des quantificateurs existentiels et universels *Exists* et *ForAll* et quelques autres symboles que nous introduirons au fur et à mesure de leur utilisation.

Un terme de RIF-BLD est soit :

- un terme simple, c'est-à-dire une constante ou une variable,
- un terme positionnel $t(t_1, \ldots t_n)$ où t est une constante et $t_1, \ldots t_n$ sont des termes,
- un terme avec arguments nommés $t(s_1 \rightarrow v_1, \ldots s_n \rightarrow v_n)$ où t est une constante et $v_1, \ldots v_n$ sont des termes et $s_1, \ldots s_n$ des noms d'argument,
- une liste de termes fermée $List(t_1 \ldots t_m)$ ou ouverte $List(t_1 \ldots t_m|t)$,
- une égalité de termes $t_1 = t_2$,
- une appartenance à une classe t_1 # t_2, où t_1 représente un objet et t_2 une classe,
- une spécialisation de classe t_1 ## t_2, où t_1 et t_2 représentent des classes,
- un frame $t[p_1 \rightarrow v_1, \ldots p_n \rightarrow v_n]$, où $p_1, \ldots p_n, v_1, \ldots v_n$ sont des termes,

– un terme externe $External(t)$ où t est un terme positionnel ou un terme avec arguments nommés.

Un terme positionnel correspond à une formule atomique en logique du premier ordre. Dans un terme avec arguments nommés, le symbole t représente un prédicat ou une fonction. Un frame représente un objet.

L'ensemble des formules atomiques de RIF-BLD comprend : les termes positionnels, les termes avec arguments nommés, les égalités de termes, les appartenances de classes, les spécialisations de classes, les frames, les termes externes. Les autres termes — constantes, variables fonctions et listes de termes — sont utilisés pour construire des formules mais ne sont pas des formules.

L'ensemble des formules de RIF-BLD comprend :

– les formules atomiques,
– les formules conjonctives $And(\varphi_1, \ldots \varphi_n)$ où $\varphi_1, \ldots \varphi_n$ sont des formules atomiques ou conjonctives ou disjonctives ou existentielles,
– les formules disjonctives $Or(\varphi_1, \ldots \varphi_n)$ où $\varphi_1, \ldots \varphi_n$ sont des formules atomiques ou des formules conjonctives ou des formules disjonctives ou des formules existentielles,
– les formules existentielles $Exist\, ?v_1, \ldots ?v_n\, (\varphi)$, où $?v_1, \ldots ?v_n$ sont des variables et où φ est une formule atomique ou conjonctive ou disjonctive ou existentielle,
– les règles d'implication φ :- ψ, où φ est une formule atomique ou une conjonction de formules atomiques non externes et où ψ est une formule atomique ou conjonctive ou disjonctive ou existentielle ; φ est la conclusion de la règle et ψ la prémisse,
– les règles universelles $Forall\, ?v_1, \ldots ?v_n\, (\varphi)$, où $?v_1, \ldots ?v_n$ sont des variables et où φ est une règle d'implication dont toutes les variables libres figurent parmi $?v_1, \ldots ?v_n$; on appelle aussi règles RIF-BLD ces règles universelles,
– les faits universels $Forall\, ?v_1, \ldots ?v_n\, (\varphi)$, où $?v_1, \ldots ?v_n$ sont des variables et où φ est une formule atomique dont toutes les variables libres figurent parmi

$?v_1, \ldots ?v_n$; les faits universels peuvent être vus comme des règles universelles sans condition,

– les groupes de règles, de faits universels, de règles d'implication sans variable libre, de formules atomiques sans variable libre et de groupes,
– les documents comportant un groupe et une séquence optionnelle de directives.

3.3.3 RIF-FLD

RIF-FLD acronyme de RIF Framework for Logic Dialect [Boley 2010d] est une recommandation du groupe de travail de RIF. RIF-FLD est conçu pour permettre de créer son propre dialecte logique, quand RIF-BLD ne couvre pas toutes les fonctionnalités nécessaires. RIF-FLD est une extension de RIF-BLD avec les termes d'agrégation, les termes distants (remote term) faisant référence à un document RIF externe et aussi les connecteurs Naf pour la négation par l'échec, Nef pour la négation.

RIF-FLD offre aussi la possibilité de créer de nouveaux termes, nouveaux connecteurs, nouveaux symboles et aussi de nouveaux quantificateurs suivant les besoins de l'utilisateur. Ainsi, RIF-FLD donne une liberté considérable aux utilisateurs pour la création de leur dialecte RIF comportant toutes les fonctionnalités requises.

3.3.4 RIF-PRD

RIF-PRD [Hawke 2013] est le dialecte de RIF adapté aux systèmes de production et aux règles métiers. C'est une extension de RIF-Core avec les fonctionnalités de mise à jour telles que la suppression, l'ajout et la modification et aussi d'action ou de fonctions externes. Une règle RIF-PRD est de la forme *if B then do H*, où B est le corps (l'antécédent) de la règle et H la tête qui est l'action à effectuer : une insertion, une suppression, une modification, un appel à une action externe ou leur

combinaison. RIF-PRD permet d'exprimer la négation, la notion de subsomption et celle d'appartenance à une classe. La Figure 3.6 présente un exemple d'utilisation de RIF-PRD.

```
Prefix(ex <http://example.com/prd>)
Forall ?customer such that ?customer#ex:Customer (
  If Not( Exists ?status
     And( ?customer[ex:status->?status]
          External(
              pred:list-contains(List("White" "Black" "Gold") ?status)) )
  Then Do( Execute(act:print(
             External(func:concat("New customer: " ?customer))))
           Assert(?customer[ex:status->"White"]) ) )
```

FIGURE 3.6 – Exemple de document RIF-PRD

3.3.5 Compatibilité de RIF avec RDF(S) et OWL

Dans le contexte du Web sémantique, la question se pose de la compatibilité de RIF avec RDF/S et OWL. L'intégration de RIF dans la pile des standards du Web sémantique et son utilisation effective pour le Web de données suppose l'écriture de règles RIF-BLD qui portent sur des données RDF et utilisent des ontologies RDFS ou OWL. Dès lors, pour assurer la compatibilité et la manipulation de données RDF, RDFS et des ontologies OWL avec RIF, le W3C définit la compatibilité de RIF-BLD/RDF et RIF-BLD/OWL dans [de Bruijn 2010], en combinant leur sémantique pour définir une interprétation commune de RIF-BLD et RDF, et RIF-BLD et OWL. Pour ce faire, trois modèles théoriques sont définis.

L'interprétation commune de RIF et RDF est définie par le couple (**I**,I) avec **I** une interprétation de RIF et I une interprétation de RDF avec la satisfaction des conditions suivantes. D'abord **I** et I doivent avoir les même domaines d'interprétation. Ensuite, les IRI et les littéraux doivent avoir la même interprétation. Enfin, les triplets et les frames sont interprétés de la même façon. L'interprétation de RDF [Hayes 2014a] comporte quatre profils différents, le profil simple pour l'interprétation et les inférences simples, le profil RDF pour les inférences RDF, le profil

RDFS pour l'interprétation et les inférence RDFS, le profil D pour l'interprétation et les inférences sur les types de données.

Pour traiter des documents RIF avec des ontologies OWL, il est nécessaire de définir l'interprétation commune de RIF et OWL. Ainsi, suivant le profil (RDF-BASED ou OWL-DIRECT) de l'ontologie utilisée nous appliquons l'interprétation correspondante RIF-OWL-Direct (utilisée pour l'interprétation d'une ontologie OWL-DL), RIF-OWL-RDF-Based pour OW-Full qui a une sémantique proche de RDF.

3.3.6 Autres dialectes RIF développés à partir de la recommandation

RIF-SILK RIF-SILK [9] est une extension de RIF-BLD et une spécialistion de RIF-FLD conformément à la recommandations RIF [Boley 2010b]. Outre les primitives de RIF-BLD, RIF-SILK supporte d'autres fonctionnalités issues du langage SILK (cf. section 3.2.4) telles que la logique d'ordre supérieur, la disjonction sur la partie conséquent d'une règle, la négation par l'échec, la négation logique, l'aggrégation. RIF-SILK propose d'autres fonctionnalités présentes dans [Grosof 2009], il s'agit des connecteurs pour les formules telles que ==> pour l'implication droite, <== pour l'implication gauche et <==> pour l'implication bidirectionnelle et la notion d'unification (:= :) et de désunification (!=) entre termes.

RIF Fuzzy Rule Dialect RIF-FRD Les travaux sur les langages de règles pour le Web sémantique s'intéressent à la logique floue [Zadeh 1965] avec la création de langages de règles floues tels que F-SWRL [Pan 2005], Vague-SWRL [Wang 2008], [Stoilos 2005]. RIF Fuzzy Rule Dialect RIF-FRD [Wang 2010] est un dialecte de RIF qui sert à échanger des règles entre systèmes de règles basés sur la logique floue. RIF-FRD est une extension de RIF-BLD avec la notion de

9. http://silk.semwebcentral.org/RIF-SILK.html

degré de vérité, les termes de RIF-FRD sont les termes correspondant à RIF-BLD auxquels on ajoute un degré de vérité.

3.4 Partage et réutilisation de règles sur le Web sémantique

Les standards du Web sémantique ont permis la création et la publication de nombreuses sources de données librement accessibles et ainsi partagées. Cet effort a permis la création du Web de données liées, un graphe global de données liées entre elles par des liens entre des ressources partagées, décrites dans différentes sources.

En complément du partage de données sur le Web de données, le partage et la réutilisation d'ontologies constitue un élément essentiel pour lier les données et faciliter le raisonnement sur les données partagées. Dès lors beaucoup de travaux se sont intéressés au partage et à la réutilisation d'ontologies sur le Web. Ceci s'est concrétisé par la publication d'ontologies sur le Web avec une réutilisation croissante. Les répertoires d'ontologies permettent de retrouver des vocabulaires publiés sur le Web. Pour faciliter l'interopérabilité et la réutilisations de vocabulaires plusieurs répertoires d'ontologies sont proposés sur le Web. Dans [d'Aquin 2012] les auteurs ont fait une étude bibliographique sur un ensemble de librairies d'ontologies existantes sur le Web pour permettre aux utilisateurs de publier et d'accéder à des ontologies. Nous citerons notamment :

- BioPortal [10] qui recense des ontologies du domaine biomédical,
- LOV [11] (Linked Open Vocabularies) qui permet de rechercher des ontologies recensées sur le Web par mots-clés,
- ONKI [12] qui permet de rechercher des ontologies par mots-clés et naviguer

10. http://bioportal.bioontology.org/
11. http://lov.okfn.org/dataset/lov/
12. http://onki.fi/

sur la base d'ontologies recensées sur le Web,

– ONTOSEARCH2 [13] qui permet de rechercher et d'interroger des ontologies sur le Web.

Les langages de représentation d'ontologie du Web sémantique entretiennent des relations étroites avec les langages de règles et les systèmes de règles. La sémantique de RDFS et celle de certains sous-ensembles de OWL - en particulier OWL 2 RL - peuvent être axiomatisées sous la forme d'implications de la logique du premier ordre qui peuvent être utilisées comme socle d'une implémentation à base de règles. De plus, le couplage d'une base de règles et d'un vocabulaire léger (en RDFS) apparaît comme une véritable alternative à un vocabulaire plus lourd (en OWL DL) et un raisonneur pour les logiques de description dont la complexité des opérations peut être élevée.

Les règles jouent ainsi de plus en plus un rôle important pour l'exploitation de la sémantique des connaissances du Web sémantique. Le Web étant un espace privilégié pour le partage et l'échange de données, il peut être naturellement envisagé pour devenir le support de publication et de partage de règles. Cependant, on constate actuellement un manque d'interopérabilité entre les systèmes de règles différents, ce qui crée des difficultés pour aligner les bases de règles pour différents systèmes et applications existants. Ainsi, plusieurs initiatives telles que RuleML, REWERSE et le groupe de travail RIF ont vu le jour pour proposer un formalisme de partage et d'échange de règles sur le Web. Ainsi de plus en plus de travaux [Boley 2007a], [Boley 2007b], [Khandelwal 2011] s'intéressent à la réutilisation de règles sur le Web.

3.4.1 L'initiative RuleML

L'iniative RuleML [14] est une organisation créée en 2000 dont l'objectif principal était de créer une famille de langages utilisée pour publier des règles sur le Web.

13. http://www.ontosearch.eu/
14. http://ruleml.org/index.html

Le groupe de travail sur RuleML vise ainsi à répondre au besoin de mise en œuvre d'un langage de règles pour le Web Sémantique afin de favoriser le partage et la réutilisation de règles sur le Web. Sa vocation était aussi de créer un standard pour permettre aux utilisateurs de systèmes d'inférences de partager des règles suite au constat de l'augmentation de l'utilisation des règles d'inférences dans le Web sémantique.

RuleML est chargé de construire un standard de règles métiers basé sur des syntaxes XML et RDF afin de favoriser l'échange et le partage de règles entre systèmes hétérogènes. RuleML couvre des familles de règles variées telles que les règles de réaction, les règles de transformation, les règles de dérivation, les faits, les requêtes et les contraintes d'intégrités. Chaque catégorie de règles de RuleML comporte une syntaxe particulière. De nombreux travaux ont été effectués sur le langage de règles RuleML, toutefois celui-ci n'a pas réussi à s'imposer comme standard de partage de règles pour le Web sémantique.

3.4.2 R2ML

R2ML [Wagner 2005] est proposé par le réseau européen d'excellence REWERSE pour permettre d'échanger des règles entre différents systèmes à base de règles. R2ML a pour objectif principal de proposer un langages d'échange de règles métier suite au constat de la croissance considérable de l'industrie de règles métier. R2ML est un langage de règles basé sur RuleML et SWRL. R2ML supporte les règles d'intégrité, les règles de dérivation, les règles de production et des règles de réaction. Il permet de sérialiser et d'échanger des règles par des outils logiciels spécifiques. Ainsi des traducteurs [15] entre R2ML et quelques langages de représentation de règles tels que F-Logic, Jess, Jena Rules, SWRL, OCL et XML sont proposés par le groupe de travail.

15. http://rewerse.net/I1/oxygen.informatik.tu-cottbus.de/rewerse-i1/@q=node_2f15.htm

3.4.3 Linked Rules

Linked Rules défini dans [Khandelwal 2011] a pour but de permettre le partage, la réutilisation, la combinaison et l'extension de règles et bases de règles. Il est basé sur les principes du Web de données. Linked Rules favorise la mise à disposition de règles et bases de règles issues de sources différentes sur le Web en les reliant entre elles. Comme le Web de données, Linked Rules est basé sur quatre principes :

- Toutes les règles et bases de règles de Linked Rules sont identifiées par des URIs.
- Lorsqu'on déréférence un URI, des informations doivent être renvoyées au format RIF.
- Les règles doivent pouvoir opérer sur des données RDF.
- Les liens entre règles et bases de règles doivent être définis.

Les règles de Linked Rules peuvent être utilisées de quatre façons différentes qui sont la réutilisation simple, l'importation de règles, la réutilisation contextuelle et la réutilisation avec contraintes. La réutilisation simple consiste à réutiliser des bases de règles pour déduire de nouvelles données, ou effectuer la traduction d'un vocabulaire vers un autre, ou vérifier si des données satisfont une base de règles. L'importation de règles consiste à référencer une base de règles donnée dans une autre base de règles. Dans ce cas les règles importées deviennent partie intégrante de la base de règles initiale. L'importation de règles est utilisée pour intégrer des règles de sources différentes.

La réutilisation contextuelle consiste à spécifier le contexte d'utilisation d'une règle. Elle consiste à spécifier les critères et contexte d'application d'une base de règles donnée. Le contexte d'utilisation peut être défini par des règles ou des données. La réutilisation avec contraintes consiste à ajouter des conditions supplémentaires pour la restriction des solutions des variables de la règle. Elle peut être utilisée pour restreindre l'application d'une règle ou base de règles (par exemple, limiter une règle définie pour tous les chercheurs aux chercheurs permanents d'In-

ria).

Il existe un ensemble de technologies et outils qui répondent aux principes de Linked Rules tel que le langage de règles AIR.

3.4.4 AIR

AIR, acronyme de Accountability In RDF, est un langage de règles pour le Web sémantique basé sur la sémantique N3Logic [Berners-Lee 2008b]. AIR permet de définir des règles de production, de la forme `if (condition) then (action-then) else (action-else)`, permettant d'exécuter l'action définie dans `action-then` si la condition a réussi, sinon effectuer l'action dans `action-else`. L'action d'une règle AIR peut consister en la création de triplets ou en l'activation d'une autre règle pouvant se trouver dans un autre document. C'est le cas de l'exemple de règle de la Figure 3.7.

Une règle AIR est identifiée par un URI unique lui permettant d'être réutilisable. Elle est décrite en utilisant les propriétés `air:if`, `air:then`, `air:else`, `air:description` and `air:assert`. Les conditions des règles AIR sont définies sous forme de patrons de graphe comme en SPARQL. Une action d'une règle AIR peut être aussi associée à une description en langage naturel.

```
@forAll :CUSTOMER .
air:if {
    :CUSTOMER a ex:customer;
      ex:status "bronze".
    @forSome :V .
    :CUSTOMER ex:purchase :V.
    :V math:greaterthan 2000. };

air:then [ air:description (
      "Si le client customer a pour statut bronze"
      "fait des achats de plus de 2000 euro alors"
      "exécuter la politique de réduction définie avec shopb:DiscountPolicy" );
    air:rule shopb:DiscountPolicy ].
```

FIGURE 3.7 – Exemple de règle AIR

AIR supporte la bibliothèque de fonctions de N3Logic et l'étend en définissant entre autres une fonction pour l'exécution de requêtes SPARQL distantes, des requête de la forme CONSTRUCT peuvent être envoyées à des points d'accès SPARQL.

AIR comprend deux ontologie : une pour la définition des règles et une autre pour la description des justifications de toutes les actions effectuées par le raisonneur, permettant de faire un débogage sur les règles et suivre les différentes étapes du raisonnement. Toutefois, les explications des actions sur certaines règles peuvent être cachées pour des raisons de confidentialité.

3.4.5 RIF Assembler

RIF Assembler [González-Moriyón 2012] est un outil qui réutilise des connaissances pour la construction automatique de bases de règles. Il permet de construire une base de règles à partir d'autres bases existantes en utilisant des métarègles pour les sélectionner. RIF Assembler exploite les annotations qu'il convertit en RDF pour attacher aux règles des métadonnées. Cette conversion est basée sur l'interprétation commune entre RIF et RDF avec la correspondance entre frames et triplets RDF. Dès lors, toutes les règles assemblées sont annotées par les métadonnées converties en RDF provenant de la source. C'est ainsi que des requêtes SPARQL peuvent être exécutées sur les annotations pour sélectionner, supprimer et utiliser des règles selon le contexte et les besoins de l'utilisateur. RIF Assembler favorise ainsi la réutilisation de règles qui est le principe de Linked Rules. RIF Assembler donne aussi la possibilité de représenter les relations entre les règles.

RIF assembler reçoit un ou plusieurs documents RIF et les regroupe en un seul document RIF. En outre, le raisonnement sur les règles est limité sur les métadonnées des règles, les règles ne contenant pas de métadonnées sont ainsi ignorées, tandis qu'en RIF les métadonnées ne sont pas obligatoires. Des règles pertinentes sont ainsi susceptibles d'être ignorées pendant cette phase de récolte des règles.

Dans notre approche nous proposons d'aller plus loin en raisonnant aussi bien sur les métadonnées que sur le contenu des règles.

3.5 Positionnement

Nous avons présenté dans ce chapitre des langages de règles existants sur le Web, nous avons décrit dans le détail le langage RIF, la recommandation du W3C pour l'échange de règles sur le Web et nous avons recensé les travaux existants sur la publication, le partage et la réutilisation de règles sur le Web sémantique. Les travaux décrits dans ce chapitre ont montré l'intérêt que porte la communauté du Web sémantique au partage et à la réutilisation de règles liées. Les ontologies sont la façon traditionnelle d'apporter de la sémantique au Web de données. En complément des données RDF, des schémas RDFS et des ontologies OWL, des règles peuvent être publiées et partagées sur le Web de données.

Dans ce contexte, le partage et la réutilisation de règles sur le Web est la problématique principale de notre travail. Dans notre approche nous considérons des bases de règles comme des sources de données, qui peuvent être publiées, partagées et interrogées sur le Web de données, permettant ainsi la sélection et la réutilisation des règles pertinentes et utiles dans un contexte ou une application particuliers. RIF, la recommandation du W3C pour le partage de règles, est jusqu'à présent peu utilisée, tandis que d'autres langages sont adoptés dans différentes communautés. Ainsi, nous avons fait le choix de travailler avec des règles écrites en SPARQL et traduites en SPIN. Notre choix s'explique par le fait que plusieurs travaux ont adopté SPARQL comme langage de règles dans la communauté du Web sémantique ; c'est notamment le cas du moteur Corese/KGRAM développé dans notre équipe. En outre la représentation de SPARQL en SPIN permet de représenter des règles dans le format RDF. Or RDF est le standard de représentation des données du Web et cette représentation permet l'interrogation du contenu des règles avec SPARQL, la recommandation du W3C pour l'interrogation de données RDF. Pour

être interopérable avec la recommandation du W3C, nous avons défini la traduction de règles SPARQL en RIF et à celle inverse de règles RIF en SPARQL. C'est l'objet du chapitre suivant.

CHAPITRE 4

Publication et partage de règles sur le Web de données

4.1 Introduction

Dans la communauté du Web sémantique, SPARQL est le standard pour l'interrogation des données RDF et plusieurs projets l'utilisent également comme langage de règles. C'est notamment le cas dans les travaux de notre équipe : le moteur Corese/KGRAM est muni d'un moteur d'inférence qui opère sur des règles représentées sous la forme de requêtes SPARQL. Dans ce contexte, et dans la perspective de l'adoption possible de la recommandation RIF du W3C comme standard pour l'échange de règles, nous nous sommes intéressés aux relations qui existent entre les langages SPARQL et RIF. Nous présentons dans ce chapitre comment traduire des requêtes SPARQL en RIF et inversement des règles RIF en SPARQL. Ce travail a précédemment fait l'objet de deux publications [Seye 2012b] [Seye 2012a].

La section 2 est consacrée à la définition du sous-ensemble de SPARQL qui peut être utilisé comme langage de règles en RIF. La section 3 présente le dialecte RIF-SPARQL, le sous-ensemble de RIF-BLD qui peut être traduit en SPARQL.

4.2 Traduction des énoncés de SPARQL en RIF-BLD

Une correspondance entre SPARQL et Datalog (nr-Datalog¬, datalog sans récursion, avec négation), le langage de requêtes et de règles des bases de données déductives, est établie dans [Polleres 2007a] et [Angles 2008]. RIF-BLD étant défini comme une extension de Datalog, nous nous sommes appuyés sur ces travaux pour établir une correspondance entre SPARQL et RIF.

Nous traduisons une requête SPARQL de la forme CONSTRUCT en règle universelle ou règle d'implication de RIF selon le schéma général suivant :

- La conclusion de la règle RIF est la conjonction de tous les termes constitués de la traduction des motifs de triplets de la clause CONSTRUCT de la requête SPARQL.
- Si la clause WHERE de la requête SPARQL ne contient que des motifs de graphe élémentaires (un motif de graphe élémentaire est un ensemble de motifs de triplet, avec éventuellement des filtres ; Un motif de triplet est formé d'un sujet, d'un prédicat et d'un objet.), alors la prémisse de la règle RIF est la conjonction de tous les termes résultant de la traduction des triplets des motifs de graphes élémentaires, avec leurs filtres s'ils en contiennent.
- Sinon, si la clause WHERE de la requête SPARQL est l'union de motifs de graphe élémentaires alors la prémisse de la règle RIF est la disjonction de tous les termes résultants des traductions de tous les motifs de graphe élémentaires, avec leurs filtres s'ils en contiennent.
- Les variables contenues dans la partie du quantificateur universel de la règle RIF sont constituées des variables résultantes de la transformation des variables et des nœuds anonymes contenus dans les motifs de graphe de la requête SPARQL.
- Si la requête SPARQL n'a pas de variable alors celle-ci est traduite en règle

d'implication de RIF (un type de formule de RIF-BLD présenté dans la section 3.3).

Nous allons maintenant donner les détails sur la traduction des énoncés contenus dans les motifs de graphe des requêtes SPARQL en termes de RIF. Notre traduction repose sur l'interprétation commune de RIF et RDF définie dans la recommandation de RIF [1]. Un triplet dans un motif de graphe SPARQL correspond à un triplet RDF avec la possibilité de variables. En considérant que s, c, c_1, c_2 sont des variables, des nœuds anonymes ou des URI, que p est une variable ou un URI, et que o est une variable, un URI, un nœud anonyme ou un littéral, les règles de transformation sont les suivantes :

- Un triplet de la forme s `rdf:type` c est traduit en une appartenance de classe $s'\#c'$.
- Un triplet de la forme (c_1 `rdfs:subClassOf` c_2) est traduit en une spécialisation de classe $c_1\#\#c_2$.
- Un ensemble de triplets représentant une liste RDF est traduit en une liste RIF fermée.
- Tout autre triplet de la forme `s p o` est traduit en un frame de la forme $s'[p'->o']$.

Dans ces triplets, les URI sont transformés en constantes de type `rif:iri`, les littéraux non typés en constantes de type `xs:string`, les littéraux typés en constantes de types correspondants, les variables et nœuds anonymes en variables.

Les opérations et fonctions définies dans les filtres sont traduites en RIF en termes externes correspondants, définis dans la bibliothèque de fonctions de RIF contenues dans la recommandation.

Soit τ_{rif} la fonction de traduction de SPARQL vers RIF-BLD. Notre proposition est résumée dans la table 4.1.

1. http://www.w3.org/TR/RIF-rdf-owl/#Interpretations

SPARQL	RIF-BLD
CONSTRUCT{H} WHERE {B}	Forall ?x_1 ... ?x_n ($\tau_{rif}(H)$:- $\tau_{rif}(B)$ avec ?x_1 ... ?x_n les variables contenues dans H et B
e_1. ... e_n.	AND($\tau_{rif}(e_1)$ ... $\tau_{rif}(e_n)$)
X rdf:type C	$\tau_{rif}(X)$ # $\tau_{rif}(C)$
C_1 rdfs:subClassOf C_2	$\tau_{rif}(C_1)$ ## $\tau_{rif}(C_2)$
s p o	τ_{rif}(s)[τ_{rif}(p) $\rightarrow$ τ_{rif}(o)]
Filter(T_1= T_2)	$\tau_{rif}(T_1)$ = $\tau_{rif}(T_2$)
(i_1 ... i_n)	List($\tau_{rif}(i_1)$,...,$\tau_{rif}(i_n)$)
?X	?X
C	C
_ : bn	?_ : bn

TABLE 4.1 – Traduction des termes SPARQL en RIF-BLD

La Table 4.2 montre trois exemples de requêtes SPARQL traduites en règles RIF-BLD.

SPARQL	RIF
CONSTRUCT {?x rdf:type ?z} WHERE {?y rdfs:subClassOf ?z . ?x rdf:type ?y}	Forall ?x ?y ?z (?x # ?z :- AND (?y ## ?z ?x # ?y))
CONSTRUCT {?x ex:uncleOf ?z} WHERE {?x ex:brotherOf ?y . ?y ex:parentOf ?z}	Forall ?x ?y ?z (?x[ex:uncleOf $\rightarrow$?z] :- AND (?x[ex:brotherOf $\rightarrow$?y] ?y[ex:parentOf $\rightarrow$?z]))
CONSTRUCT {?x rdf:type ex:Adult} WHERE {?x rdf:type ex:Person . ?x ex:age ?y FILTER(?y >= 18)}	Forall ?x ?y ?z (?x # ex:Adult :- AND (?x # ex:Person ?x[ex:age $\rightarrow$?y] External(pred:numeric-greater-than-or-equal(?y 18))))

TABLE 4.2 – Traduction de requêtes SPARQL en règles RIF-BLD

En outre, une base de règles SPARQL peut être accompagnée d'un graphe de données RDF auxquelles elles s'appliquent et d'une base d'opérations INSERT DATA en considérant SPARQL 1.1 Update. Ces données et opérations peuvent être traduites en conjonctions de formules RIF-BLD selon le même schéma : tous les triplets *s p o* de la base RDF et des graphes opérandes des opérations INSERT DATA sont traduits en conjonctions de frames $s'[p' -> o']$.

Notre traduction s'applique sur les requêtes SPARQL dont les clauses WHERE sont constituées de motifs de graphe élémentaire (basic graph pattern) ou d'unions de motifs de graphes élémentaires. Par conséquent, les requêtes contenant des motifs complexes impliquant les opérateurs OPTIONAL, MINUS, NOT EXISTS, des chemins de prorpiétés ou des sous-requêtes sont exclus de la traduction de SPARQL en RIF.

4.3 Traduction des énoncés de RIF-BLD en SPARQL

L'intégration de RIF dans la pile des standards du Web sémantique et son utilisation effective pour le Web de données suppose l'écriture de règles RIF-BLD qui portent sur des données RDF et utilisent des ontologies RDFS ou OWL. Un scénario typique est l'échange de règles RIF avec des graphes RDF et des ontologies RDFS ou OWL entre systèmes à base de règles capables de prendre en compte des ontologies et d'opérer sur des bases de faits RDF. Un autre scénario est celui où des règles RIF-BLD peuvent être prises en compte dans un moteur de recherche SPARQL pour répondre à des requêtes en retrouvant non seulement des faits présents dans un graphe de données RDF mais aussi d'autres faits qui peuvent être déduits du graphe à l'aide de ces règles. Les moteurs sémantiques qui implémentent la sémantique de RDFS ou de OWL opèrent des inférences, par exemple pour retrouver toutes les ressources d'un certain type (en retrouvant également celles déclarées

avec des sous-types).

La comptabilité de RIF-BLD avec RDF et OWL [de Bruijn 2010] fait de ce dialecte le plus adapté pour les moteurs sémantiques existants.

Traduire les règles RIF-BLD en SPARQL permet de porter RIF-BLD sur les moteurs qui opèrent sur des règles représentées en SPARQL (requêtes de la forme CONSTRUCT), notamment le moteur sémantique Corese développé dans notre équipe.

Pour caractériser le dialecte RIF qui peut être représenté en SPARQL, nous avons cherché à traduire chacun des énoncés de RIF-BLD en SPARQL. Pour cela, une première étape préliminaire a consisté à normaliser les énoncés de RIF-BLD en les réduisant en clauses de Horn afin de diminuer la complexité des transformations. Un sous-ensemble d'énoncés RIF-BLD s'est alors avéré simplement traduisible en SPARQL. Nous avons ensuite cherché à augmenter au maximum ce sous-ensemble.

4.3.1 Normalisation des formules de RIF-BLD

Nous normalisons les énoncés de RIF-BLD en reposant sur les travaux de [Grimm 2007] qui effectuent la conversion d'une ontologie WSML en règles Datalog et ceux de [Marte 2011] qui proposent une conversion de RIF-BLD en règles Datalog. Cette normalisation nous permet de simplifier la mise en correspondance de RIF-BLD avec SPARQL ; la traduction des énoncés de RIF-BLD en SPARQL est alors la composition de transformations élémentaires appliquées récursivement.

Cette normalisation est présentée dans les tables 4.3 et 4.4. La table 4.3 montre le résultat de l'application de la fonction de normalisation τ_{nf} aux différents types de formules RIF-BLD. La table 4.4 montre le résultat de l'application de la fonction de normalisation τ_{lt} aux règles RIF-BLD pour produire des règles plus simples. Elle supprime notamment les conjonctions dans la tête (conséquent) des règles à traduire et la disjonction dans le corps (antécédent) des règles. Dans ces

deux tables, R est un ensemble de formules de RIF-BLD; F, F_i et G_i expriment des formules de RIF-BLD ; T, P, V constituent des termes de base, C et S des constantes, *?X* une variable. H dénote une formule atomique, B une formule conditionnelle, S et G des groupes (Group).

Les normalisations numéro 2 et 3 suppriment les occurrences de conjonctions et disjonctions sur un argument. Les normalisations numéro 4 et 5 suppriment les imbrications de conjonctions et disjonctions inutiles. Les normalisations 6 et 7 appliquent la loi distributive aux formules conjonctives. Les normalisations 8 et 9 appliquent la normalisation à toutes les formules d'une conjonction et disjonction. La normalisation numéro 10 permet de transformer un frame d'arguments multiples en une conjonction de frames d'un argument. La règle de normalisation numéro 11 applique la normalisation à toutes les règles d'un ensemble de règles. Les règles de normalisation numéro 12 et 13 transforment des règles dont le conséquent est une conjonction de formules en un ensemble de règles dont le conséquent est une formule atomique. Les règles de normalisation numéro 14 et 15 transforment une règle dont l'antécédent est une disjonction en un ensemble de règles pour supprimer la disjonction sur le corps d'une règle.

	Expression RIF-BLD	Expression RIF-BLD normalisée
1	$\{ F_1 \dots F_n \}$	$\{ \tau_{nf}(F_1), \dots, \tau_{nf}(F_n) \}$
2	$AND(F)$	$\tau_{nf}(F)$
3	$OR(F)$	$\tau_{nf}(F)$
4	$AND(F_1 \dots F_i$ $AND(G_1 \dots G_m)$ $F_{i+2} \dots F_n$	$AND(F_1 \dots F_i$ $G_1 \dots G_m$ $F_{i+2} \dots F_n)$
5	$OR(F_1 \dots F_i$ $OR(G_1 \dots G_m)$ $F_{i+2} \dots F_n)$	$OR(F_1 \dots F_i$ $G_1 \dots G_m$ $F_{i+2} \dots F_n)$

6	$AND(F_1 \dots F_i$ $OR(G_1 \dots G_m)$ $F_{i+2} \dots F_n)$	$AND(F_1 \dots F_{i-1}$ $\tau_{nf}(OR(AND(F_i\ G_1) \dots AND(F_i\ G_m)))$ $F_{i+2} \dots F_n)$
7	$AND(OR(G_1 \dots G_m)$ $F_1 \dots F_n)$	$AND(\tau_{nf}(OR(AND(G_1\ F_1) \dots$ $AND(G_m\ F_1)))$ $F_2 \dots F_n)$
8	$AND(F_1 \dots F_n)$	$AND(\tau_{nf}(F_1) \dots \tau_{nf}(F_n))$
9	$OR(F_1 \dots F_n)$	$OR(\tau_{nf}(F_1) \dots \tau_{nf}(F_n))$
10	$T[P_1 \rightarrow V_1 \dots P_n \rightarrow V_n]$	$AND(T[P_1 \rightarrow V_1] \dots T[P_n \rightarrow V_n])$

TABLE 4.3: Normalisation des expressions de RIF-BLD par la fonction τ_{nf}

	Règle RIF-BLD	Règle RIF-BLD normalisée
11	$R_1 \dots R_n$	$\{\tau_{lt}(R_1), \dots \tau_{lt}(R_n)\ \}$
12	$Forall\ ?X_1 \dots ?X_m$ $AND(H_1 \dots H_n)$:- B	$\{\ \tau_{lt}(Forall\ ?X_1 \dots ?X_m\ H_1$:- $B), \dots,$ $\tau_{lt}(Forall\ ?X_1 \dots ?X_m\ H_m$:- $B)\ \}$
13	$Forall\ ?X_1 \dots ?X_m$ H :- $OR(B_1 \dots B_n)$	$\{\ \tau_{lt}(Forall\ ?X_1 \dots ?X_m\ H$:- $B_1),$ $\dots \tau_{lt}(Forall\ ?X_1 \dots ?X_m\ H$:- $B_n)\ \}$
14	$AND(H_1 \dots H_n)$:- B	$\{\ \tau_{lt}(\ H_1$:- $B), \dots \tau_{lt}(\ H_n$:- $B)\ \}$
15	H :- $OR(B_1 \dots B_n)$	$\{\ \tau_{lt}(H$:- $B_1), \dots \tau_{lt}(H$:- $B_n)\ \}$
16	H :- B	$(\tau_{nf}(H)$:- $\tau_{nf}(B))$
17	$Forall\ ?X_1 \dots ?X_m\ H$:- B	$Forall\ ?X_1 \dots ?X_m$ $(\tau_{nf}(H)$:- $\tau_{nf}(B))$

TABLE 4.4: Normalisation des règles de RIF-BLD par la fonction τ_{lt}

4.3.2 Le dialecte RIF-SPARQL

Pour porter RIF-BLD sur les moteurs de règles qui opèrent sur des règles représentées en SPARQL, nous avons cherché à définir le sous-ensemble maximal du dialecte RIF-BLD qui peut être exprimé dans le langage SPARQL.

RIF-BLD est une extension de Datalog. Dans [Angles 2008], une équivalence est établie entre l'expressivité de Datalog non récursive avec négation et SPARQL, et les auteurs ont proposé une traduction des énoncés de SPARQL vers Datalog et vice versa. En nous inspirant de ces travaux, nous proposons une traduction des énoncés de RIF-BLD en SPARQL. Cette intégration est basée sur la traduction de l'arbre de syntaxe abstraite d'un document RIF-BLD en arbre de syntaxe abstraite de SPARQL. Toutes les formules de RIF-BLD, règles d'implication et règles universelles, sont traduites en requêtes CONSTRUCT de SPARQL. L'antécédent de la règle RIF-BLD est traduit en motif de graphe et le conséquent de la règle en un ensemble de motifs de triplets. Les formules atomiques, conjonctives, disjonctives et existentielles sont traduites en SPARQL comme suit :

- une formule atomique est traduite comme décrit dans les tables Table 4.5 et Table 4.6;
- une formule conjonctive est traduite en motif de graphe élémentaire;
- une formule disjonctive est traduite en l'union de motifs de graphes élémentaires;
- une formule existentielle est traduite en traduisant la formule qu'elle contient et en ignorant le quantificateur existentiel.

La fonction de traduction des frames de RIF-BLD traduit une appartenance de classe et spécialisation de classe vers des motifs de graphe élémentaires de SPARQL. Soit τ_{SPARQL}: $R_{RIF} \mapsto R_{sp}$, R_{RIF} représente un énoncé de RIF-BLD et R_{sp} son correspondant en SPARQL. τ_{SPARQL} est l'inverse de τ_{rif} (cf. section 4.2) .

Nous commençons d'abord par présenter la traduction des énoncés de RIF-BLD traduisibles simplement en SPARQL. Nous introduisons notre proposition de

transformation des termes (terme de position, terme à arguments nommés, termes d'égalité) qui n'ont pas de correspondant direct en SPARQL. Les types de données, les fonctions et les prédicats prédéfinis de RIF-BLD seront représentés par leur correspondant direct en SPARQL en utilisant les filtres de SPARQL.

4.3.2.1 RIF-SPARQL Core

Nous considérons les énoncés de RIF-BLD dont l'interprétation commune avec des énoncés de RDF(S) et OWL par conséquent de SPARQL est définie dans [de Bruijn 2010]. Dans cette traduction présentée dans la Table 4.5, T, T_1, T_2, T_n P, V dénotent des termes simples (constantes ou variables) de RIF-BLD et E_1, E_2 des constantes, des variables ou des fonctions externes de RIF-BLD et C une constante. Notre proposition est résumée dans la Table 4.5.

RIF-BLD	SPARQL
Forall $?X_1 \ldots ?X_n$ (H :- B)	CONSTRUCT{τ_{SPARQL}(H)} WHERE {τ_{SPARQL}(B)}
(T[P $\rightarrow$ V])	τ_{SPARQL}(T) τ_{SPARQL}(P) τ_{SPARQL}(V)
T_1 # T_2	$\tau_{SPARQL}(T_1)$ rdf:type $\tau_{SPARQL}(T_2)$
T_1 ## T_2	$\tau_{SPARQL}(T_1)$ rdfs:subClassOf $\tau_{SPARQL}(T_2)$
$E_1 = E_2$	Filter($\tau_{SPARQL}(E_1)$ = $\tau_{SPARQL}(E_2)$)
External(C($T_1 \ldots T_n$))	Filter($\tau_{SPARQL}(C)(\tau_{SPARQL}(T_1) \ldots \tau_{SPARQL}(T_n))$)
List(T_1,...,T_n)	($\tau_{SPARQL}(T_1) \ldots \tau_{SPARQL}(T_n)$)
$?X$	$?X$
C	C

TABLE 4.5 – Traduction des termes de RIF-BLD en SPARQL : RIF-SPARQL Core

4.3.2.2 RIF-SPARQL

Pour étendre RIF-SPARQL Core, nous nous sommes intéressés à la traduction en SPARQL des termes positionnels et des termes à arguments nommés de RIF-BLD. Nous les traduisons en SPARQL par des motifs de graphe élémentaire. Notre

proposition est résumée dans la table 4.6, où C et N sont des constantes et V et T des termes simples. Nous avons défini un vocabulaire RDF identifié par le préfixe de son espace de noms *rs*. La propriété *rs:arity* (le nombre d'arguments) est introduite pour garder la sémantique des termes. En effet, *f(x)* n'est pas une conséquence logique de *f(x,y)*, alors que l'absence de la propriété *rs:arity* conclurait une implication logique entre *f(x,y)* et *f(x)*.

RIF-BLD	SPARQL
C($N_1 \rightarrow V_1 \ldots N_n \rightarrow V_n$)	_:b_n a `rs:NamedArgs` _:b_n `rs:name` C _:b_n `rs:arity` n _:b_n `rs:`N_1 $\tau_{SPARQL}(V_1)$. ... _:b_n `rs:`N_n $\tau_{SPARQL}(V_n)$.
C($T_1 \ldots T_n$)	_:b_n a rs:Positional _:b_n `rs:name` C _:b_n `rs:arity` n _:b_n `rs:`arg_1 $\tau_{SPARQL}(T_1)$. ... _:b_n `rs:`arg_n $\tau_{SPARQL}(T_n)$.

TABLE 4.6 – Traduction des termes positionnels et termes à arguments nommés de RIF-BLD en SPARQL : RIF-SPARQL

En résumé en comparaison avec RIF-BLD, la restriction principale de RIF-SPARQL est l'exclusion des faits universels, des listes ouvertes et la limitation aux termes de base à des constantes et variables. Les termes de base sont utilisés pour construire d'autres termes. Les termes de position et les listes ne sont pas admis comme termes de base en RIF-SPARQL. Nous présentons dans la Figure 4.1 la grammaire du dialecte RIF-SPARQL en notation EBNF pour la syntaxe de présentation.

Règles:

```
RULE        ::= CLAUSE |
                'Forall' Var+ '(' CLAUSE ')'
CLAUSE      ::= ATOMIC1 ':-' FORMULA |
                'AND' '(' ATOMIC* ')' ':-' FORMULA
FORMULA     ::= ATOMIC |
                ('AND' | 'OR') '(' FORMULA* ')' |
                'Exists' Var+ '(' FORMULA ')'
ATOMIC1     ::=  Atom | Frame | Member | Subclass
ATOMIC      ::=  ATOMIC1 | Equal
Atom        ::= UNITERM
UNITERM     ::= Const '(' TERM1* ')' |
                Const '(' (Name '->' TERM1)*) ')'
Equal       ::= TERM4 '=' TERM4
Member      ::= TERM1 '#' TERM1
Subclass    ::= TERM1 '##' TERM1
Frame       ::= TERM1 '[' (TERM1 '->' TERM2)* ']'
TERM1       ::= Const | Var
TERM2       ::= TERM1 | List
TERM4       ::= TERM1 | 'External' '(' Expr ')'
Expr        ::= UNITERM
List        ::= 'List' '(' TERM1* ')'
Const       ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Var         ::='?'Name
```

Faits:

```
FACT        ::= Atom1 | Frame1 | Member1 | Subclass1
Atom1       ::= UNITERM1
UNITERM1    ::= Const '(' Const* ')' |
                Const '(' (Name '->' Const)* ')'
Member1     ::= Const '#' Const
Subclass1   ::= Const '##' Const
Frame1      ::= Const '[' (Const '->' Const)* ']'
```

FIGURE 4.1 – Grammaire EBNF de RIF-SPARQL

Les faits, c'est-à-dire les formules atomiques sans variables, peuvent être traduits en SPARQL par des opérations INSERT DATA, à l'exception de ceux contenant des termes à égalité et des termes externes.

4.4 Conclusion

Dans ce chapitre, nous avons défini un dialecte de RIF qui peut être traduit en SPARQL, que nous appelons RIF-SPARQL. C'est un sous-ensemble de RIF-BLD qui exclut les listes ouvertes, les termes d'égalité, les arguments non simples (les listes, les termes de position et les externes) et les règles récursives. Le travail d'implémentation que nous avons réalisé sur le moteur Corese et les résultats que nous obtenons sur la base de tests publiée par le W3C sont présentés dans le chapitre 5 de ce mémoire.

Nous avons fait le choix du langage SPIN/RDF pour partager des règles sur le Web de données. SPARQL est un des langages utilisés pour représenter des règles. De plus, le partage de règles sous syntaxe SPIN facilite leur réutilisation sur le Web de données du fait de la syntaxe RDF ; en effet elles peuvent ainsi être considérées comme des sources de données. Ainsi SPIN permet de travailler dans un format unique pour représenter les données, les classes, les propriétés et les règles d'un domaine. En outre, la représentation en SPIN/RDF des règles permet de les relier entre elles, de les annoter en RDF et d'interroger à la fois leurs annotations et leurs contenus en utilisant le langage SPARQL, le standard pour l'interrogation de données RDF.

Dans la suite de cette thèse, nous nous intéressons à l'exploitation de bases de règles représentées en RDF. Le travail de traduction de RIF en SPARQL et de SPARQL en RIF présenté dans ce chapitre assure que nos travaux permettent aussi bien de manipuler des règles exprimées dans le standard RIF, en se limitant au dialecte RIF-SPARQL que nous avons défini.

CHAPITRE 5

Sélection et réutilisation de règles sur le Web

5.1 Introduction

La réutilisation de connaissances partagées est au cœur du Web de données. Publier des règles SPARQL en SPIN/RDF permet de les partager, et les réutiliser en les interrogeant avec le standard d'interrogation de données SPARQL. SPIN permet en outre de lier la représentation du contenu des règles avec d'éventuelles méta-données et avec d'autres données du Web. En particulier, les règles en SPIN peuvent être directement liées avec des données RDF(S) ou des ontologies OWL. Ces *données* RDF peuvent ensuite être interrogées avec des requêtes SPARQL pour sélectionner des règles intéressantes à réutiliser dans un contexte spécifique.

Dans [González-Moriyón 2012] les auteurs ont exploré la réutilisation de règles RIF et proposé l'outil RIF Assembler qui permet de sélectionner et réutiliser des règles RIF en interrogeant les méta-données de ces règles, en les traduisant en RDF selon l'interprétation commune de RIF et RDF. Cependant, en RIF, les méta-données ne sont pas obligatoires et les règles non annotées ne peuvent pas être réutilisées avec RIF Assembler. De plus, seules les méta-données sont exploitées dans cette approche, et pas le contenu des règles.

Ce chapitre décrit notre approche *générale* et *unifiée* de la réutilisation de règles SPARQL représentées en SPIN/RDF. Nous proposons une approche de la réutilisation de règles à partir de sources de règles existantes en sélectionnant des règles

par leur contenu ainsi que les méta-données qui leur sont associées. Ce travail a précédemment été publié dans [Seye 2014].

La section 2 présente la sélection de règles pertinentes basée sur l'interrogation des méta-données associées aux règles. La section 3 décrit la sélection de règles pertinentes basée sur l'interrogation du contenu des règles. La section 4 présente la validation de bases de règles par rapport à une ontologie. La section 5 aborde la mise à jour de bases de règles.

5.2 Sélection basée sur l'interrogation des méta-données associées aux règles

Le Web est un espace global de publication d'informations, où la crédibilité du producteur de l'information constitue un critère important dans la recherche d'informations. Dans certains domaines, il est primordial de cibler des critères qui peuvent apporter des éléments de réponse les plus adaptés possible. Par exemple dans le milieu médical, il est préférable d'interroger des sources officielles, en France l'Agence Nationale de Sécurité du Médicament et des produits de santé (ANSM) pour les informations sur des médicaments. En outre dans ce milieu (médical), les informations (ici on s'intéresse aux règles) les plus récentes sont en général celles les plus pertinentes. Et ces informations sont inscrites en général au niveau des méta-données.

Les méta-données permettent de mettre à disposition une documentation de référence relative aux informations publiées sur le Web. Par conséquent, les méta-données des règles peuvent contenir des informations importantes sur les règles telles que la source, l'auteur, le titre, le sujet, etc. La recommandation du W3C, RIF, suggère d'utiliser Dubling Core, des propriétés de RDFS et OWL pour annoter les règles, spécifiquement : `owl:versionInfo`, `rdfs:isDefinedBy`, `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `dc:creator`, `dc:description`, `dc:date`, et

`dc:maker`.

Le modèle Open Annotation [1] est également adapté pour l'annotation de règles, qui fait une distinction entre le corps (`oa:body`) et le but (`oa:target`) d'une annotation de ressource, et permet ainsi de distinguer méta-données et description de contenu. La notion de sélecteur (`oa:Selector`) permet de décrire séparément différentes parties d'une ressource et permet, dans le cas d'une règle, de distinguer des annotations portant sur sa prémisse ou sa conclusion.

La réutilisation de règles publiées sur le Web basée sur les méta-données permet de répondre au scénario où pour traiter une source de données RDF particulière, un utilisateur souhaite sélectionner des règles selon l'organisme qui les publie, ou bien selon leur date de publication, ou leur sujet, c'est-à-dire, plus généralement, selon des critères qui peuvent être attachés aux règles dans des méta-données.

Par exemple, prenons le scénario où l'ANSM publie des informations sur l'évaluation des médicaments sur le Web de données, dont certaines sont représentées par des règles SPARQL, et où un utilisateur recherche des règles sur le benfluorex [2] dans les données de l'ANSM. Par exemple les affirmations de L'ANSM sur le benfluorex suivantes « `l'utilisation de médicaments contenants le benfluorex cause des risques d'atteinte des valves cardiaques` » et « `l'autorisation de mise sur le marché des spécialités contenant du benfluorex est suspendue` » peuvent être formalisées comme suit :

```
Règle 1 :
  PREFIX  ex: <http://www.example.org>
  PREFIX  drug: <http://www.example.org/drug#>
  CONSTRUCT { ?patient ex:risque ex:Anomalie_valvulaire }
  WHERE {
    ?drug ex:contains drug:Benfluorex .
```

1. http://www.openannotation.org/spec/core/
2. http://www.ema.europa.eu/ema/index.jsp?curl=pages/medicines/human/referrals/Benfluorex/human_referral_000220.jsp

```
    ?patient ex:uses ?drug
  }

Règle 2 :
  PREFIX  ex: <http://www.example.org>
  PREFIX  drug: <http://www.example.org/drug#>
  CONSTRUCT { ?drug a drug:unauthorizedDrug }
  WHERE { ?drug ex:contains drug:Benfluorex }
```

Pour publier ces règles sur le Web, il serait intéressant de spécifier dans des méta-données associées qu'elles sont définies par l'ANSM qui est une source officielle. Pour ce faire, il s'agit d'identifier les règles par des URI et de les décrire en RDF. Notamment, l'URI identifiant une règle serait le sujet d'une propriété `dc:source` et la valeur un URI désignant l'ANSM, et le sujet d'une propriété `dc:subject` dont la valeur désignerait le benfluorex. La requête SPARQL de la Figure 5.1 permet de rechercher des règles (`sp:Construct`) sur le benfluorex parmi les données publiées par l'ANSM.

```
PREFIX sp:   <http://spinrdf.org/sp#>
PREFIX drug: <http://www.example.org/drug#>
PREFIX dc:   <http://purl.org/dc/terms/>
SELECT ?x
WHERE {
  ?x a sp:Construct .
  ?x dc:source <https://icrepec.afssaps.fr/Public> .
  ?x dc:subject drug:benfluorex
}
```

FIGURE 5.1 – Sélection de règles selon leur annotation

5.3 Sélection de règles basée sur l'interrogation de leur contenu

Publier des règles SPARQL en SPIN permet de lier la représentation de leur contenu avec d'autres données du Web. En particulier, elles sont liées avec des don-

nées RDF(S) ou des ontologies OWL qui peuvent être utilisées lors de la sélection de règles par interrogation de leurs contenus. Un exemple de scénario de réutilisation de règles publiées sur le Web par le contenu est celui relatif à la prise en compte des connaissances de domaine lors de l'exploitation d'une source de données RDF, où l'utilisateur recherche des règles de domaine relatives à ces données. Il recherche donc sur le Web, parmi toutes les règles d'inférence publiées, celles dont le contenu référence des ressources qu'il souhaite exploiter. En considérant un ensemble de données RDF(S) et/ou OWL, les règles qui utilisent le même vocabulaire sont susceptibles de s'appliquer à ces données. Sélectionner ainsi des règles par leur contenu permet d'identifier les règles pertinentes pour un jeu de données.

Par exemple, la requête SPARQL de la Figure 5.2 permet de sélectionner toutes les règles dont la prémisse utilise la propriété `ex:hasUncle`.

```
PREFIX sp: <http://spinrdf.org/sp#>
PREFIX ex: <http://www.example.org/humans#>
SELECT DISTINCT ?x
WHERE {
  ?x a sp:Construct
  ?x sp:where ?m
  ?m (! sp:void)+ ?s
  ?s sp:predicate ex:hasUncle
}
```

FIGURE 5.2 – Sélection des règles par le contenu

5.3.1 Recherche de règles d'un domaine

Comme le montre l'exemple précédent, la sélection de règles permet de chercher des règles en se basant sur leur contenu, sur les classes et propriétés utilisées. En analysant le contenu des règles on peut déterminer si la base de règles concerne un domaine. Cette analyse peut être réalisée en faisant l'appariement des classes et propriétés utilisées dans la construction des règles avec les classes et propriétés utilisées dans le vocabulaire du domaine.

Par exemple, la requête SPARQL de la Figure 5.3 permet de sélectionner toutes les règles dont la prémisse contient des propriétés représentant des relations familiales. Cette requête exploite la sémantique du schéma RDFS utilisé dans l'écriture des règles de la base et permet de retrouver toutes celles dont la prémisse contient des propriétés représentant des relations familiales. Cela permet de récupérer par exemple les règles contenant la propriété `ex:hasUncle` en exploitant la sémantique du schéma, à savoir la relation de subsomption entre les propriétés `ex:hasUncle` et `ex:hasFamilyRelationShip`.

```
PREFIX sp: <http://spinrdf.org/sp#>
PREFIX ex: <http://www.example.org/humans#>
SELECT DISTINCT ?x
WHERE {
  ?x a sp:Construct
  ?x sp:where ?m
  ?m (! sp:void)+ ?s
  ?s sp:predicate ?z
  ?z rdfs:subPropertOf* ex:hasFamilyRelationShip
}
```

FIGURE 5.3 – Sélection des règles par le contenu, exploitant une ontologie de domaine

Il peut également être intéressant de sélectionner des règles dans le contenu desquelles apparaissent des classes ou des propriétés appartenant à une ontologie donnée. En effet, pour un domaine donné, les règles qui utilisent le vocabulaire de ce domaine peuvent contenir des informations intéressantes permettant aux agents logiciels de raisonner avec les données de ce domaine. La requête de la Figure 5.4 permet de sélectionner les règles qui utilisent des classes et propriétés appartenant à un schéma RDFS chargé. Pour ce faire, il faut interroger à la fois la représentation SPIN/RDF des règles et l'ontologie. La sous-requête définie entre les lignes 3 et 9 interroge l'ontologie et sélectionne les classes et propriétés ; le patron de graphe défini entre les lignes 11 et 14 interroge les règles et sélectionne celles dont la prémisse utilise une classe ou propriété de l'ontologie.

```
0   PREFIX sp: <http://spinrdf.org/sp#>
1   SELECT ?x
2   WHERE {
3     { SELECT DISTINCT ?resource
4       WHERE {
5         { ?resource rdf:type rdfs:Class }
6         UNION
7         { ?resource rdf:type rdf:Property }
8       }
9     }
10
11    ?x a sp:Construct .
12    ?x (! sp:void)+ ?s .
13    ?s ?p ?resource
14    VALUES ?p { sp:subject sp:predicate sp:object }
15  }
```

FIGURE 5.4 – Sélection des règles d'un domaine

5.3.2 Calcul du graphe de dépendance entre règles

En programmation logique, l'application d'une règle peut être déclenchée par l'application d'autres règles, si ces dernières produisent des faits qui les rendent applicables. Par conséquent, les règles qui peuvent être déclenchées par des règles directement applicables aux données sont également pertinentes pour ces données. Pour déterminer l'ensemble de règles pertinentes pour une source de données, il faut donc considérer les relations de dépendance entre règles. Pour optimiser l'algorithme de chaînage avant, dans [Baget 2004], les auteurs ont défini la notion de graphe de dépendances entre règles. Les nœuds du graphe sont des règles et les arêtes relient une règle à celles dont elle dépend. Une règle R_i dépend d'une règle R_j si l'application de R_j sur la base de données considérée est susceptible de déclencher celle de R_i. La sélection de R_j comme règle pertinente doit entrainer la sélection de R_i.

Par exemple considérons la base de connaissances suivante :

```
Ontologie :
@prefix  ex: <http://example.org/>.
ex:Thesard rdfs:subClassOf  ex:Doctorant.
```

```
ex:Doctorant rdfs:subClassOf  ex:Thesard.

Données :
@prefix  ex: <http://example.org/>.
ex:Lily rdf:type  ex:Doctorant.
ex:Max rdf:type  ex:Thesard.

Règle 1 :
CONSTRUCT {?x a ?c1}
WHERE {
  ?c1 owl:equivalentClass ?c2 .
  ?x a ?c2
}

Règle 2 :
CONSTRUCT { ?c1 owl:equivalentClass ?c2 }
WHERE {
  ?c1 rdfs:subClassOf ?c2 .
  ?c2 rdfs:subClassOf ?c1
}
```

La règle 1 ne s'applique pas aux données car le motif de triplet `?c1 owl:equivalentClass ?c2` ne s'apparie pas aux données. Toutefois, l'application de la règle 2 produit le triplet `ex:Doctorant owl:equivalentClass ex:Thesard`, rendant l'application de la règle 1 possible. Donc la règle 1 dépend de la règle 2 car l'application de la règle 1 est déclenchée par celle de la règle 2.

Le calcul du graphe de dépendances entre règles est basé sur la confrontation de la condition d'une règle et de la conclusion d'une autre règle. Dans [Baget 2009], le calcul de dépendance se fait par test d'unification entre la conclusion d'une règle et les conditions des autres règles. Ce calcul de dépendance peut également être vu comme une sélection de règles. Dans ce cas, les règles pertinentes sont celles qui dépendent d'une règle considérée. Etant données des règles représentées en SPARQL, le calcul de dépendance est basé sur la correspondance entre

le patron de graphe de la clause CONSTRUCT d'une règle et celui de la clause WHERE des autres règles de la base. Soit R_i et R_j deux règles SPARQL de la forme CONSTRUCT, R_i dépend de R_j, s'il y a une correspondance entre un des triplets de la clause WHERE de R_i avec un des triplets de la clause CONSTRUCT de R_j. Soient $t_1 = (s_1\ p_1\ o_1)$ et $t_2 = (s_2\ p_2\ o_2)$ deux motifs de triplet, t_1 est unifiable à t_2 si les trois conditions suivantes sont remplies :

1. p_1 est une variable ou p_2 est une variable ou $p_1 = p_2$
2. s_1 est une variable ou un nœud anonyme ou s_2 est une variable ou un nœud anonyme ou $s_1 = s_2$
3. o_1 est une variable ou un nœud anonyme ou o_2 est une variable ou un nœud anonyme ou $o_1 = o_2$

La représentation des règles en format SPIN permet d'implémenter le calcul du graphe de dépendance par une requête SPARQL. Par exemple, la requête de la Figure 5.3.2 calcule le graphe de dépendances entre règles de la sémantique de RDFS. En effet, RDFS et OWL permettent de raisonner avec des données RDF en définissant des relations entre classes et entre propriétés à l'aide d'un ensemble de constructeurs. La sémantique de RDFS et celle de OWL 2 RL peuvent être axiomatisées sous forme de règles. Patrick Hayes et ses coauteurs [Hayes 2014b] ont défini la sémantique de RDFS sous forme de triplets axiomatiques et de règles d'inférences.

Considérons par exemple la règle rdfs9, qui stipule que si un graphe RDF contient les triplets `(X rdfs:subClassOf Y)` et `(C rdf:type X)` alors le triplet `(C rdf:type Y)` est inféré. Nous formalisons cette règle par la requête SPARQL suivante :

```
CONSTRUCT { ?c rdf:type ?y }
WHERE {
  ?x rdfs:subClassOf ?y .
  ?c rdf:type ?x .
}
```

La représentation en SPIN de cette règle pourra être interrogée avec la requête de la figure 5.5. Appliquée à chacune des règles d'une base de règles, cette requête permet de calculer le graphe des dépendances qui existent entre ces règles.

Description de la requête 5.5

La sous-requête définie entre les lignes 6 et 15 permet de sélectionner les règles (avec la variable `?r1`), les motifs de triplets de leur clause CONSTRUCT (désignée par la variable `?t1` valeur de la propriété `sp:templates`), et le sujet, le prédicat et l'objet d'un modèle des motifs de triplets, désignés par les valeurs des propriétés `sp:subject`, `sp:predicate` et `sp:object`. Le filtre de la ligne 13 permet de sélectionner les seuls motifs de triplets faisant intervenir des propriétés primitives de RDFS ou `rdf:type`.

Les lignes 16 et 17 permettent de sélectionner les clauses WHERE (avec la variable `?w`) des autres règles (désignées par la variable `?r2`). Le patron de graphe entre les lignes 18 et 26 permet de restreindre la sélection des règles désignées par `?r2` en testant la correspondance entre les motifs de triplets de la clause CONSTRUCT des règles désignées par `?r1` et les motifs de triplets de la clause WHERE des règles désignées par `?r2`.

Le patron de graphe défini entre les lignes 0 et 4 permet de construire un graphe RDF représentant le graphe de dépendances de règles avec la propriété `sp:related`, le triplet `?r1 sp:related ?r2` indique que `?r2` dépend de `?r1`.

La fonction `st:apply-templates-with` utilisée aux lignes 27 et 28 appelle le pretty-printer du moteur Corese/KGRAM pour produire le contenu des règles dans la syntaxe concrète de SPARQL.

5.3.3 Détection de règles transitives

La représentation des règles en format SPIN permet de rechercher des règles selon différentes caractéristiques. Par exemple, la requête SPARQL de la Figure

```
0  CONSTRUCT {
1    ?r1 sp:related ?r2
2    ?r1 sp:value  ?pp1
3    ?r2 sp:value  ?pp2
4  }
5  WHERE {
6    { SELECT *
7      WHERE {
8        ?r1 sp:templates ?t1
9        ?t1 rdf:rest*/rdf:first
10         [ sp:subject ?s1 ;
11           sp:predicate ?p1 ;
12           sp:object ?o1 ]
13       FILTER(strstarts(?p1, rdfs:)||(?p1=rdf:type))
14     }
15   }
16   ?r2 sp:where ?w
17   FILTER(?r1 != ?r2)
18   FILTER EXISTS {
19     ?w rdf:rest*/rdf:first
20     [ sp:subject ?s2 ;
21       sp:predicate ?p2 ;
22       sp:object ?o2 ]
23       FILTER(isBlank(?s1) || isBlank(?s2) || ?s1 = ?s2)
24       FILTER(isBlank(?p1) || isBlank(?p2) || ?p1 = ?p2)
25       FILTER(isBlank(?o1) || isBlank(?o2) || ?o1 = ?o2)
26   }
27   BIND (st:apply-templates-with(st:spin, ?r1) as ?pp1)
28   BIND (st:apply-templates-with(st:spin, ?r2) as ?pp2)
29 }
```

FIGURE 5.5 – Requête de calcul du graphe de dépendance entre règles de la sémantique de RDFS

5.6 permet de sélectionner les règle transitives.

5.3.4 Optimisation du raisonnement basée sur la sélection de règles

Les requêtes de sélection de règles présentées dans les sections 5.3.1, 5.3.2 et 5.6 peuvent être vues comme des étapes préliminaires d'un algorithme de chaînage avant permettant d'en réduire le coût. Celui-ci augmente avec la taille de la base de règles et le raisonnement avec de grandes bases de règles nécessite d'optimiser cet algorithme. Beaucoup de travaux se sont intéressés à ce problème ; c'est notamment le cas de l'algorithme de RETE [Forgy 1982] et d'algorithmes basés sur le graphe de dépendances entre règles [Baget 2004].

Une première optimisation consiste à ne sélectionner que les règles dans le contenu desquelles apparaissent des classes ou propriétés appartenant à l'ontologie des données auxquelles on souhaite appliquer les règles. Dans le cas de sources de données de grande taille, un tel filtrage peut devenir crucial pour la faisabilité des inférences. La requête de la Figure 5.4 permet de sélectionner les seules règles qui utilisent des termes appartenant à un schéma RDFS chargé.

Une seconde optimisation est la sélection des seules règles susceptibles de s'appliquer à une source de données. Dans le cas où les données n'utilisent qu'un petit nombre des classes et propriétés d'une ontologie, cette sélection peut permettre un gain de temps important. Les règles sélectionnées sont celles susceptibles de s'appliquer directement aux données, c'est-à-dire dont la description contient des classes ou propriétés présentes dans les données, et celles qui en dépendent. Le calcul consiste donc à déterminer les règles qui s'appliquent à une source de données puis de leur ajouter celles qui en dépendent, en utilisant la requête de la figure 5.5.

Un cas particulier est la sélection des règles de la sémantique du modèle RDFS ou OWL pertinentes pour la source de donnée considérée. Ainsi, dans le cas d'une ontologie ne comportant que des classes primitives, la plupart des règles de la sé-

```
SELECT ?q
WHERE {
  ?q a sp:Construct ;
    sp:templates (
   [ sp:subject ?x ;
     sp:predicate ?p ;
     sp:object ?z ] ) .
  {
  ?q sp:where (
   [ sp:subject ?x ;
     sp:predicate ?p ;
     sp:object ?y ]
   [ sp:subject ?y ;
     sp:predicate ?p ;
     sp:object ?z ] )
  }
  UNION {
  ?q sp:where (
   [ sp:subject ?p ;
     sp:predicate rdf:type ;
     sp:object owl:TransitiveProperty]
   [ sp:subject ?x ;
     sp:predicate ?p ;
     sp:object ?y ]
   [ sp:subject ?y ;
     sp:predicate ?p ;
     sp:object ?z ] )
  }
}
```

FIGURE 5.6 – Détection de règles transitives

mantique de RDFS ou de OWL deviennent inutiles, c'est-à-dire ne s'appliquent pas, et l'économie de la tentative de leur application devient précieuse dans le cas d'une source de données de grande taille.

Un moteur en chaînage avant fonctionne par saturation en bouclant sur la base de règles tant que de nouveaux faits peuvent être déduits. Les règles transitives ont la particularité d'être applicables (i.e. leur exécution est susceptible de produire de nouveaux résultats) tant qu'elles produisent de nouveaux résultats. Elles ont tendance à augmenter le nombre d'itérations. Partant de ce constat, Olivier Corby a effectué une optimisation du moteur de règles Corese/KGRAM basé sur la sélection de règles transitives, pour les traiter de manière particulière. L'optimisation consiste à exécuter à saturation les règles transitives dans une boucle dédiée.

5.4 Validation de bases de règles

Des règles sélectionnées à partir du Web peuvent être invalides ou devenir invalides. La validation de bases de règles permet de chercher s'il n'existe pas d'anomalies dans une base de règles. Dans [Zacharias 2008], Valentin Zacharias a défini un ensemble d'anomalies que peut comporter une base de règles. Il définit les anomalies comme des parties ou des propriétés dans une base de règles qui sont, avec une forte probabilité, le résultat d'un défaut. Ces anomalies permettent de souligner des incohérences possibles dans la définition d'une base de règles reposant sur une ontologie. Les différents types d'anomalies définis sont la contradiction, la redondance, l'insatisfiabilité d'une condition de règle, l'utilisation de classes non définies, l'utilisation de propriétés non définies ....

La validation d'une base de règles par rapport à une ontologie consiste à analyser la base de règles afin de savoir si elle est compatible avec l'ontologie en question. Nous proposons une approche de la validation d'une base de règle qui repose sur un ensemble de requêtes SPARQL de la forme ASK qui testent différents types d'anomalies dans la représentation SPIN/RDF des règles. Nous supposons

que toutes les classes et propriétés utilisées dans les données RDF sur lesquelles seront appliquées les règles sont définies dans l'ontologie.

5.4.1 Classe non définie

Une règle viole un domaine si elle utilise dans sa construction une classe qui n'est pas définie dans le vocabulaire du domaine en question. Détecter une classe non définie consiste à chercher dans le contenu des règles s'il existe une classe qui n'apparait pas dans l'ontologie du domaine.

Exemple :

La règle 1 de cet exemple utilise les classes `ex:Lecturer` et `ex:Course` qui ne sont pas définies dans l'ontologie du domaine des relations familiales. Par conséquent cette règle n'est pas valide par rapport à cette ontologie.

La requête de la figure 5.8 permet de chercher s'il existe une classe utilisée dans la définition d'une règle qui n'est pas déclarée dans l'ontologie des données. Le motif de graphe entre les lignes 2 et 5 permet de sélectionner des règles utilisant dans leur construction une classe : il s'agit des règles SPIN pour lesquelles `rdf:type` est une valeur de la propriété `sp:predicate`. Le motif de graphe représenté entre les lignes 7 et 9 permet de détecter les règles `?x` utilisant une classe `?resource` non définie dans l'ontologie chargée.

5.4.2 Propriété non définie

Une règle viole un domaine si elle utilise dans sa construction une propriété qui n'est pas définie dans le vocabulaire du domaine en question. Détecter une propriété non définie consiste à chercher dans le contenu des règles s'il existe une propriété qui n'apparait pas dans l'ontologie du domaine.

Exemple, en considérant l'ontologie des relations familiales utilisée dans l'exemple précédent :

```
Ontologie :
 @prefix ex: <http://example.org/> .
 ex:Person rdf:type rdfs:Class .
 ex:Man rdf:type rdfs:Class .
 ex:Woman rdf:type rdfs:Class .
 ex:hasParent rdf:type rdf:property .
 ex:hasUncle rdf:type rdf:property .
 ex:hasBrother rdf:type rdf:property .

Règle 1 :
  PREFIX ex: <http://example.org/>
  CONSTRUCT { ?x rdf:type ex:Lecturer }
  WHERE { ?x rdf:type ex:Person }

Règle 2 :
  PREFIX ex: <http://example.org/>
  CONSTRUCT { ?x rdf:type ex:Person }
  WHERE { ?x rdf:type ex:Man }

Règle 3 :
  PREFIX ex: <http://example.org/>
  CONSTRUCT { ?x rdf:type ex:Person }
  WHERE { ?x rdf:type ex:Woman }
```

```
0   PREFIX sp: <http://spinrdf.org/sp#>
1   ASK {
2     ?x a sp:Construct .
3     ?x (! sp:void)+ ?s .
4     ?s sp:predicate rdf:type ;
5        sp:object  ?resource .
6     FILTER (! isBlank(?resource))
7     FILTER NOT EXISTS {
8       ?resource rdf:type ?class
9       VALUES ?class {rdfs:Class  owl:Class}
10    }
11  }
```

FIGURE 5.7 – Requête de détection de l'existence de règles avec une classe non définie

```
Règle 1 :
PREFIX ex: <http://example.org/>
CONSTRUCT {?x ex:hasUncle ?z}
WHERE {
  ?x ex:hasParent ?y.
  ?y ex:hasBrother ?z
}

Règle 2 :
PREFIX ex: <http://example.org/>
CONSTRUCT { ?z ex:hasRelation ?y }
WHERE { ?z ex:hasParent ?y }
```

La règle 2 utilise la propriété `ex:hasRelation` qui n'est pas définie dans l'ontologie du domaine des relations familiales. Par conséquent cette règle n'est pas valide par rapport à cette ontologie.

La requête de la figure 5.8 permet de chercher s'il existe une propriété utilisée dans la définition d'une règle qui n'est pas déclarée dans l'ontologie des données.

Le motif de graphe entre les lignes 2 et 4 permet de sélectionner les propriétés utilisées dans la construction des règles SPIN ; ce sont les valeurs de la propriété `sp:predicate`. La ligne 5 `FILTER (! isBlank(?predicate))` permet de ne pas considérer les motifs de triplet dont le prédicat est une variable, car un motif dont le prédicat est une variable ne peut pas créer une incohérence (en SPIN, les variables sont représentées par des blank nodes). Le motif de graphe représenté entre

```
0   PREFIX sp: <http://spinrdf.org/sp#>
1   ASK {
2     ?x a sp:Construct .
3     ?x (! sp:void)+ ?s.
4     ?s sp:predicate ?predicate.
5     FILTER (! isBlank(?predicate))
6     FILTER NOT EXISTS {
7       ?predicate rdf:type  ?property
8       VALUES ?property {
9         rdf:Property owl:ObjectProperty owl:DatatypeProperty }
10    } }
```

FIGURE 5.8 – Requête de détection de l'existence de règles avec une propriété non définie

les lignes 6 et 9 permet de détecter les règles `?x` utilisant une propriété `?property` non définie dans l'ontologie chargée.

5.4.3 Détection d'incohérence

La détection de classes et/ou de propriétés non définies dans la construction des règles SPIN peut être intéressante pour l'utilisateur. Cette anomalie peut être liée à trois cas principaux :

- cas 1: le concepteur a oublié de déclarer des concepts du domaine. Dans ce cas, détecter les classes et propriétés non définies aidera à compléter l'ontologie.
- cas 2 : l'ontologie du domaine a évolué. Comme les règles sont liées aux classes et propriétés modélisées dans l'ontologie, la modification de l'ontologie peut avoir un impact sur les règles, et par conséquent causer des problèmes de cohérence. Dans ce cas l'utilisateur pourra réadapter ses règles à la version actuelle de l'ontologie.
- cas 3 : la base de règles n'est pas pertinente pour l'ontologie. Dans ce cas l'utilisateur peut tout simplement éliminer la base de règles en question ou en supprimer les règles qui ne sont pas pertinentes pour ses données.

Pour détecter des règles non pertinentes, nous sélectionnons les règles qui utilisent des classes et/ou des propriétés non définies dans le vocabulaire. Pour ce faire nous utilisons la requête de la Figure 5.9. Elle repose sur les mêmes principes que les deux requêtes précédentes.

5.5 Mise à jour de bases de règles

SPARQL 1.1 Update [3] permet d'exprimer des opérations pour effectuer des mises à jour sur un graphe RDF. En représentant en SPIN une base de règles,

3. http://www.w3.org/TR/sparql11-update/

```
0  PREFIX sp: <http://spinrdf.org/sp#>
1  SELECT DISTINCT ?x  ?predicate  ?resource
2  WHERE {
3    ?x a sp:Construct .
4    ?x (! sp:void)+ ?s.
5      {
6        ?s  sp:predicate ?predicate.
7        FILTER (! isBlank(?predicate))
8        FILTER NOT EXISTS {
9          ?predicate rdf:type ?property
10         VALUES ?property {
11           rdf:Property owl:ObjectProperty owl:DatatypeProperty }
12     }
13     UNION
14     {
15       ?s sp:predicate rdf:type;
16          sp:object ?resource .
17       FILTER (! isBlank(?resource))
18       FILTER NOT EXISTS {
19         ?resource rdf:type ?class
20         VALUES ?class { rdfs:Class owl:Class }
21       }
22     }
23   }
24 }
```

FIGURE 5.9 – Requête de détection de règles invalides

le contenu et/ou les métadonnées des règles peuvent être mis à jour en utilisant SPARQL Update. Il peut s'agir de modifier les règles, par exemple en les généralisant, en remplaçant une classe ou une propriété par une classe ou une propriété plus générale ; de supprimer des règles ; d'ajouter des annotations. Comme décrit dans le chapitre précédent, nous recommandons d'associer à chaque règle un graphe nommé. La mise à jour d'une règle correspond ainsi à la mise à jour du graphe nommé dans lequel elle est définie.

Dans la section précédente nous avons montré comment détecter des problèmes d'incohérence des règles avec l'ontologie des données. Certaines de ces incohérences peuvent être causées par l'évolution de l'ontologie. Les règles qui présentent une incohérence par rapport à l'ontologie peuvent être supprimées en utilisant une opération DELETE. La requête 5.10 permet de supprimer les règles qui font référence à une propriété non définie dans l'ontologie.

Il est également possible de propager certains changements de l'ontologie dans la base de règles en utilisant SPARQL Update afin de corriger certaines incohérences. Dans le cas d'un changement qui consiste à renommer des classes ou des propriétés de l'ontologie associée, des opérations de mise à jour peuvent être utilisées pour modifier les occurrences des classes et propriétés changées dans la représentation SPIN des règles conformément à l'ontologie. Par exemple, l'opération DELETE/INSERT de la Figure 5.11 remplace la propriété `ex:hasMother` par la propriété la plus générale `ex:hasParent` dans toute les règles.

```
PREFIX sp: <http://spinrdf.org/sp#>
DELETE {
  GRAPH ?graph { ?s ?p ?o } }
WHERE {
  GRAPH ?graph {
           ?x a sp:Construct .
           ?x (! sp:void)+ ?m.
           ?m  sp:predicate ?predicate.
           FILTER (! isBlank(?predicate))
           FILTER NOT EXISTS {
             ?predicate rdf:type ?property
             VALUES ?property {
               rdf:Property owl:ObjectProperty owl:DatatypeProperty }
           }
           ?s ?p ?o
  }
}
```

FIGURE 5.10 – Exemple d'opération de suppression de règle

```
PREFIX sp: <http://spinrdf.org/sp#>
PREFIX ex: <http://www.example.org/humans#>
DELETE {
  GRAPH ?graphrule {
    ?s sp:predicate ex:hasMother.
  }
}
INSERT {
  GRAPH ?graphrule {
    ?s  sp:predicate ex:hasParent.
  }
}
WHERE {
  GRAPH ?graphrule {
    ?s sp:predicate ex:hasMother.
  }
}
```

FIGURE 5.11 – Exemple de requête de mise à jour de contenu de règle

Un autre cas d'utilisation de la mise à jour de bases de règles est l'adaptation de règles au domaine. Nous considérons le cas où un utilisateur travaille avec l'ontologie `FOAF`. En considérant la correspondance entre `foaf:Person` et `dbpedia:Person`, cet utilisateur peut réutiliser des règles portant sur `dbpedia:Person`. Ainsi, il pourrait sélectionner les règles portant sur

`dbpedia:Person` et remplacer toutes les occurrences de `dbpedia:Person` par `foaf:Person`.

Enfin, un autre scénario est l'ajout d'annotations sur les règles pour mieux les décrire. Cette fonctionnalité peut être implémentée par une opération INSERT. L'exemple de la Figure 5.12 annote les règles relatives aux relations familiales en spécifiant leur auteur, avec la propriété `dc:creator`.

```
PREFIX ex: <http://example.org/>
PREFIX sp: <http://spinrdf.org/sp#>
PREFIX dc: <http://purl.org/dc/terms/>
INSERT {
  GRAPH ?graphrule {
    ?x  dc:creator ex:Olivier . }
}
WHERE {
  GRAPH ?graphrule {
    ?x a sp:Construct
    ?x (! sp:void)+ ?s
    ?s sp:predicate ?z
    ?z rdfs:subPropertOf* ex:hasFamilyRelationShip
  }
}
```

FIGURE 5.12 – Exemple de requête de mise à jour des annotations de règle

5.6 Conclusion

Dans ce chapitre nous avons montré l'intérêt de publier des règles de façon unifiée dans le modèle des données du Web. En effet, la représentation de règles en SPIN/RDF permet de sélectionner des règles par leur contenu ou leur annotation, de faire de la validation de base de règles par rapport à un modèle (ontologie), d'effectuer des mises à jour sur les règles et aussi de propager certains changements d'ontologie vers les règles associées en se basant sur les modèles et techniques du Web de données. Toutes ces opérations peuvent être implémentées par des requêtes SPARQL Query et Update.

Cependant, l'approche est générale et nos travaux sur RIF permettraient de

considérer aussi bien cet autre modèle de règles. L'essentiel est en effet de disposer d'une syntaxe RDF du langage de règles considéré permettant l'interrogation du contenu des règles et le traitement uniforme du contenu et des annotations de règles. Notre travail présenté dans le chapitre sur la définition d'un dialecte RIF-SPARQL et une fonction de traduction d'énoncés RIF en SPARQL, combinée avec la compilation de requêtes SPARQL en RDF permet de le ramener à ces conditions.

Le moteur Corese/KGRAM permet d'implémenter les différents scénarios d'interrogation de sources de règles SPIN/RDF. Nous allons présenter dans le chapitre suivant plusieurs scénarios typiques de réutilisation de règles avec des données liées que nous avons expérimentés sur plusieurs ensembles de données.

CHAPITRE 6

Implémentations et expérimentations

6.1 Introduction

Ce chapitre décrit le travail d'implémentation et d'évaluation des différentes contributions de cette thèse. Il repose sur le moteur sémantique Corese/KGRAM [1] qui permet d'interroger des sources de données RDF et d'appliquer des règles d'inférence sur celles-ci.

La section suivante introduit le moteur sémantique Corese/KGRAM. La section 3 est consacrée à l'implémentation de RIF-SPARQL et aux résultats que nous avons obtenus sur la base de test publiée et approuvée par le groupe de travail du W3C sur RIF. La section 4 décrit une implémentation des scénarios de sélection de règles. La section 5 décrit une implémentation de l'optimisation du moteur de règles basée sur le calcul de graphe de dépendance.

6.2 Corese/KGRAM

Corese/KGRAM [Corby 2004] [Corby 2010] [Corby 2012] est une plateforme pour le Web sémantique. Elle est conçue pour l'interrogation de graphes de connaissances quelconques et en particulier de graphes RDF. Elle intègre un moteur de règles d'inférence et un moteur de règles de transformation [Corby 2014].

1. http://wimmics.inria.fr/corese

6.2.1 Le moteur de recherche sémantique Corese/KGRAM

Corese est l'acronyme de COnceptual REsource Search Engine [Corby 2004]. C'est un moteur sémantique pour l'interrogation de graphes RDF, basé sur une représentation interne et des mécanismes de raisonnement sur un modèle de graphe générique. KGRAM est l'acronyme de Knowledge Graph Abstract Machine, c'est une reconception modulaire de Corese conçu pour l'interrogation de graphes de connaissances quelconques et en particulier de graphes RDF qui peuvent être distribués sur différentes sources. Il permet d'interroger des graphes RDF(S) avec le langage de requêtes SPARQL. Il implémente la sémantique de RDFS et OWL 2 RL et le langage de requête SPARQL 1.1 [Harris 2013] et permet donc d'interroger des données RDF avec des requêtes SPARQL Query mais aussi d'effectuer des mises à jour sur les données RDF avec des opérations SPARQL Update (INSERT, DELETE, INSERT DATA, DROP ...).

Corese/KGRAM repose sur une interface `Producer` pour l'énumération de triplets RDF. Dans le cas où les données sont réparties, un `Metaproducer` itère sur les sources disponibles et énumère, en parallèle et de manière transparente, des triplets provenant de ces sources. Finalement, une interface d'invocation à distance permet la fédération de sources de données RDF distribuées sur le Web de données [Corby 2012]. Ainsi, ces extensions font de Corese/KGRAM un moteur adapté au Web de données. La Figure 6.1 résume l'architecture du moteur Corese/KGRAM.

6.2.2 Le moteur de règles de KGRAM

Le moteur Corese/KGRAM intègre deux moteurs de règles, l'un en chaînage avant et l'autre en chaînage arrière, et un parser (analyseur syntaxique) de règles représentées en SPARQL. Il permet ainsi d'effectuer des inférences sur une base RDF(S) pour répondre à des requêtes. Les règles sont vues comme partie intégrante d'une ontologie, complétant les déclarations RDFS ou OWL.

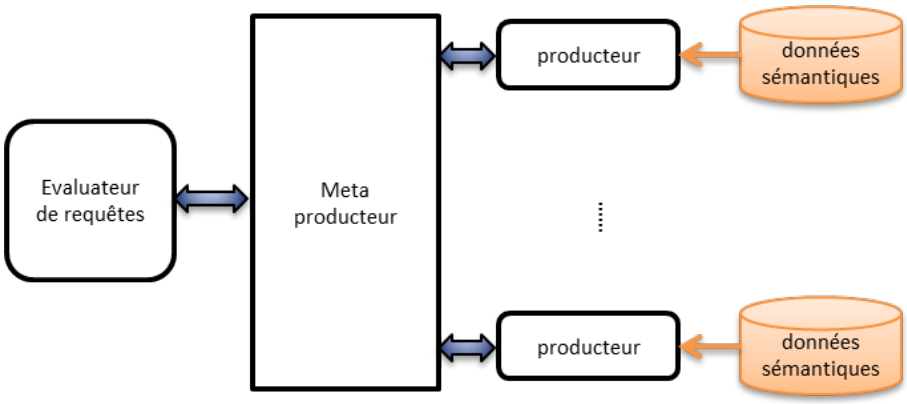

FIGURE 6.1 – Producteurs de KGRAM

Corese/KGRAM peut utiliser son moteur d'inférence pour raisonner sur des données distribuées sur plusieurs sources. Il peut ainsi raisonner sur des données distribuées avec une base de règles SPARQL centralisée. C'est ce que montre la Figure 6.2.

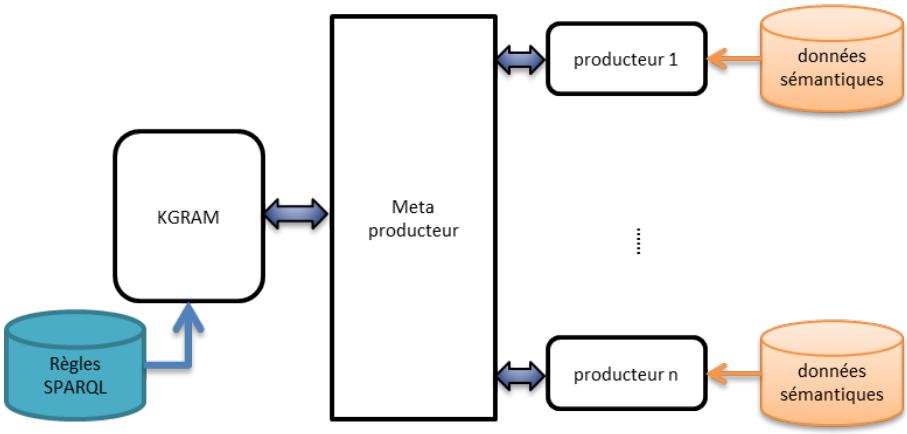

FIGURE 6.2 – Base de règles SPARQL centralisée avec des données distribuées

En outre, chaque producteur de KGRAM peut être implémenté par une instance de KGRAM, et donc être muni à la fois d'un moteur de recherche et d'un moteur d'inférences ce qui permet de raisonner sur des données liées distribuées avec une base de règles SPARQL pour chaque source de données RDF. C'est ce que montre la Figure 6.3. Ce scénario fait sens lorsqu'un ensemble de règles est spécifique à un

jeu de données, mais peut également s'appliquer sur d'autres sources de données.

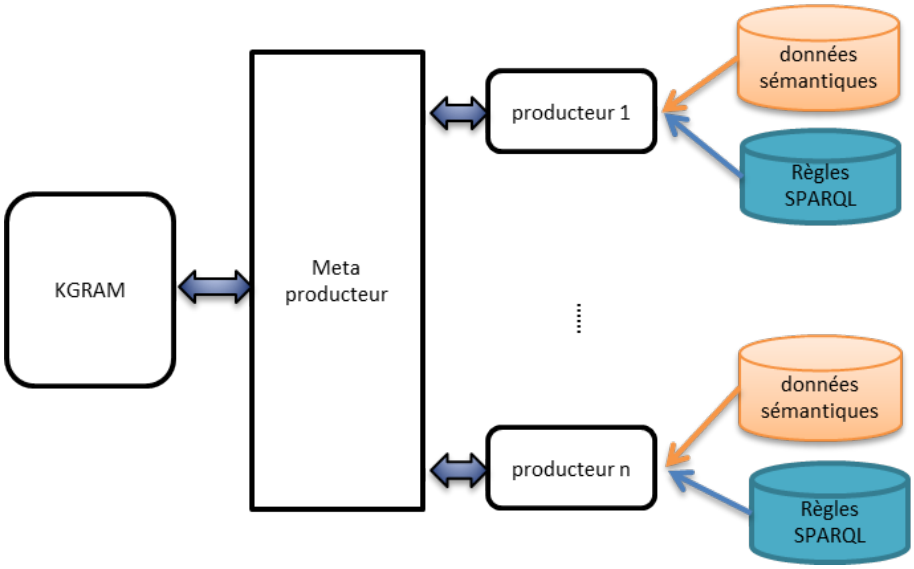

FIGURE 6.3 – Sources de données RDF distribuées avec leurs bases de règles SPARQL associées

Ainsi le moteur de règles de Corese/KGRAM répond aux différents scénarios de distribution des données du Web et est donc bien adapté pour raisonner sur le Web de données.

6.2.3 Le moteur de transformation de Corese/KGRAM

Corese/KGRAM intègre un moteur de transformation [Corby 2014] pour produire des données dans la syntaxe concrète de leur langage de représentation à partir de leur arbre de syntaxe abstraite au format RDF. En particulier, cinq bases de règles de transformation sont définies pour les langages OWL, Turtle, SPIN, SQL et HTML.

Corese/KGRAM est donc muni d'un moteur de transformation pour produire des requêtes dans la syntaxe concrète de SPARQL à partir de la représentation en SPIN. Il est également muni d'un parser pour produire des représentations en SPIN de règles données dans la syntaxe concrète de SPARQL. Ainsi, les utilisa-

teurs de Corese/KGRAM n'ont pas besoin d'apprendre le langage de règles SPIN, mais peuvent utiliser le langage standard SPARQL. La Figure 6.4 montre les trois instructions Java permettant de donner une requête SPARQL sous la forme d'une chaîne de caractères (`query`) et d'obtenir la représentation en SPIN de cette requête dans une nouvelle chaîne de caractères (`spin`).

```
String query = " PREFIX ex: <http://www.example.org/humans#>"
           +"CONSTRUCT { ?x ex:hasUncle ?z .}"
           +"WHERE {"
           +"?x ex:hasParent ?y ."
           +"?y ex:hasBrother ?z .}";
SPINProcess sp = SPINProcess.create();
String spin = sp.toSpin(query);
```

FIGURE 6.4 – Traduction de requêtes SPARQL en SPIN avec Corese

La richesse de Corese nous a permis de mettre en œuvre notre approche du partage et de la réutilisation de règles sur le Web de données avec peu de développement.

6.3 Implémentation de RIF-SPARQL avec le moteur Corese/KGRAM

Cette section présente l'implémentation du dialecte RIF-SPARQL avec les moteurs de règles SPARQL de Corese/KGRAM. Cette implémentation est basée sur l'ensemble des correspondances que nous avons établies entre RIF et SPARQL dans la section 4.3 et la traduction de l'arbre de syntaxe abstraite d'un document RIF-BLD en arbre de syntaxe abstraite de SPARQL en conservant la sémantique des énoncés.

La Figure 6.5 présente l'architecture générale de notre implémentation. Un analyseur syntaxique de RIF-BLD, qui produit un arbre de syntaxe abstraite de RIF-BLD. Un traducteur transforme un sous-arbre de l'AST de RIF-BLD correspondant au dialecte RIF-SPARQL en l'AST de SPARQL de Corese/KGRAM.

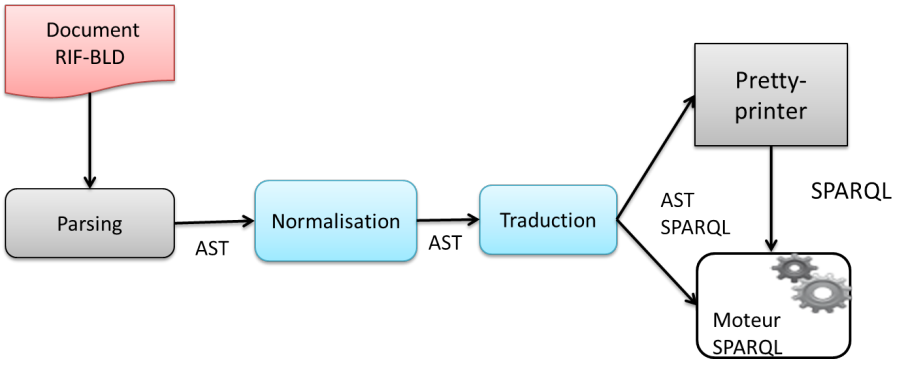

FIGURE 6.5 – Architecture de l'implémentation de RIF-SPARQL

6.3.1 Analyse syntaxique de RIF-BLD

A partir de la conversion normative de la syntaxe de présentation de RIF-BLD vers sa syntaxe XML fournie dans la recommandation de RIF, un modèle d'arbre de syntaxe abstraite a été développé dans notre équipe pour être partagé entre la syntaxe de présentation et la syntaxe XML de RIF-BLD. Un analyseur syntaxique (parser) pour chacune de ces syntaxes de RIF-BLD est développé. L'architecture générale de l'analyseur est présentée dans la figure 6.6.

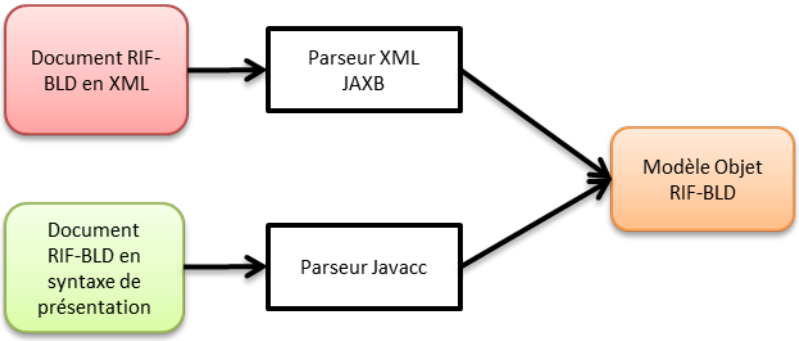

FIGURE 6.6 – Architecture de l'analyseur syntaxique de RIF-BLD

6.3.2 Traduction de l'AST RIF-BLD vers l'AST de SPARQL

Nous avons effectué une traduction partielle de l'arbre de syntaxe abstraite d'un document de RIF-BLD en l'arbre de syntaxe abstraite SPARQL défini dans le moteur Corese/KGRAM. Le sous-arbre de l'AST de RIF-BLD que nous traduisons correspond au dialecte RIF-SPARQL que nous avons défini dans la section 4.3. Le principe de notre algorithme consiste à parcourir l'AST de RIF en entrée (stocké dans un objet Java) et à appliquer récursivement à ses nœuds une méthode de traduction pour créer de nouveaux objets Java qui sont des nœuds de l'AST SPARQL en sortie.

6.3.3 Implémentation du régime d'inférence RIF-SPARQL

Dans cette section nous présentons notre implémentation de RIF-SPARQL avec le moteur sémantique Corese/KGRAM. Notre travail consiste à faire supporter les implications logiques de RIF-BLD par Corese/KGRAM. Nous utilisons les transformations qui suivent.

"INSERT DATA" pour construire ou enrichir le graphe de données RDF : les faits contenus dans la base de connaissances de RIF-BLD sont ajoutés au graphe de données RDF avec des opérations INSERT DATA. Un nouveau graphe RDF est construit s'il n'en existait pas déjà un.

"CONSTRUCT WHERE" pour construire la base de règles : les règles universelles contenues dans la base de connaissances de RIF-BLD sont traduites en requêtes SPARQL de la forme CONSTRUCT. Ces requêtes sont les règles d'inférence qui seront appliquées au graphe RDF.

"ASK" pour vérifier les buts : pour vérifier une implication logique entre le graphe de données RDF et un but, celui-ci est représenté par une requête SPARQL

de la forme ASK. L'implication logique est vérifiée si l'exécution de la requête ASK avec le moteur Corese/KGRAM ayant chargé le graphe de données RDF et la base de règles SPARQL renvoie vrai.

Considérons l'exemple du document RIF tiré de la base de test d'inférences positives sur RIF proposée par le W3C [2] de la Fig. 6.7 .

```
Document (
Prefix(ppl <http://example.org/people#>)
Prefix(tax <http://example.org/scientific-classification#>)
Prefix(cpt <http://example.org/concepts#>)
Group (
  Forall ?C ?I ?P ?V (?I[?P -> ?V] :- And(?C[?P->?V] ?I # ?C))
  ppl:john # cpt:Person
  cpt:Person[tax:phylum -> tax:Chordata] )
)
```

FIGURE 6.7 – Exemples d'un document de RIF-BLD

Ce document contient une règle universelle et deux faits. D'abord nous effectuons la traduction des espaces de nommage (namespace) du document RIF vers SPARQL. Les deux faits sont traduits en opérations INSERT DATA pour enrichir le graphe RDF. La règle universelle est traduite en requête SPARQL de la forme CONSTRUCT. Nous traduisons ainsi ce document comme suit.

2. http://www.w3.org/2005/rules/test/repository/tc/Classification-inheritance/Classification-inheritance-premise.rifps

RIF-BLD	SPARQL
Prefix(ppl <http://example.org/people#>) Prefix(tax <http://example.org/scientific-classification#>) Prefix(cpt <http://example.org/concepts#>)	Prefix ppl: <http://example.org/people#> Prefix tax: <http://example.org/scientific-classification#> Prefix cpt: <http://example.org/concepts#>
ppl:john # cpt:Person	INSERT DATA { ppl:john rdf:type cpt:Person . }
cpt:Person[tax:phylum -> tax:Chordata]	INSERT DATA { cpt:Person tax:phylum tax:Chordata . }
Forall ?C ?I ?P ?V (?I[?P -> ?V] :- And (?C[?P -> ?V] ?I#?C))	CONSTRUCT {?I ?P ?V } WHERE { ?C ?P ?V . ?I rdf:type ?C . }

La formule RIF-BLD à prouver, contenue dans le document Classification-inheritance-conclusion.rifps [3], est le frame ppl:john[tax:phylum -> tax:Chordata]. Cette formule est transformée en la requête SPARQL ASK {ppl:john tax:phylum tax:Chordata}, pour tester si elle est la conséquence logique de la base de connaissances. Le triplet

```
ppl:john tax:phylum tax:Chordata
```

sera ajouté à la base grâce à une opération INSERT DATA. Par conséquent l'évaluation de la requête

```
ASK { ppl:john tax:phylum tax:Chordata }
```

renverra la valeur vraie (true).

En chaînage arrière, l'évaluation de la requête

```
ASK { ppl:john tax:phylum tax:Chordata }
```

invoque la règle CONSTRUCT qui déclenche l'exécution d'une requête

3. http://www.w3.org/2005/rules/test/repository/tc/Classification-inheritance/Classification-inheritance-conclusion.rifps

```
ASK { ppl:john rdf:type cpt:Person.
      cpt:Person tax:phylum tax:Chordata }
```

qui dans l'exemple renverra la valeur true.

L'implication logique entre les connaissances du document et le but `ppl:john[tax:phylum -> tax:Chordata]` est ainsi établie.

Un document RIF-BLD peut importer un ou plusieurs documents RIF-BLD de même que des documents RDF, RDFS et des ontologies OWL en précisant leur emplacement et profil. Pour gérer la prise en compte des documents importés dans la base de connaissances, nous étudions le profil de chaque document importé et effectuons le traitement adapté suivant le profil. Si le document importé est de type RIF-BLD le profil ne sera pas précisé, dans ce cas les inférences que nous allons effectuer prendront en compte les prémisses du document importé. Si le document est un document RDF, RDFS ou une ontologie OWL nous appliquons les principes de l'interprétation commune de RIF-BLD et RDF, RDFS ou OWL définis dans [de Bruijn 2010] pour la prise en compte des données contenues dans ces documents dans les inférences.

6.3.4 Evaluation de l'implémentation de RIF-BLD

Description de l'expérimentation

Dans cette section nous décrivons les résultats obtenus en exécutant la correspondance de RIF-BLD et SPARQL sur le moteur de chaînage arrière de Corese/KGRAM. Pour évaluer notre méthode, nous avons utilisé la base de test publiée et approuvée par le groupe de travail de RIF [W3C 2005] pour l'implication logique. La base de test comporte deux grandes familles de cas de test : l'implication positive avec les cas de test pour lesquels les conclusions des règles sont les conséquences logiques des prémisses et l'implication négative avec des cas de test pour lesquels les conclusions de la base de test ne sont pas des conséquences logiques des prémisses. Chaque cas comporte un document où sont définis les faits

et règles et un document pour la conclusion (formule à vérifier) et de façon optionnelle des documents importés.

Pour vérifier les implications logiques entre les prémisses et les conclusions, nous appliquons les transformations décrites dans la section 6.3.3.

Résultat

La réussite de la traduction d'un cas de test est indiquée par le terme 'réussi'. Les cas de test non traités sont indiqué par 'non traité', il s'agit des cas de test non couverts par le dialecte RIF-SPARQL.

num	Cas de test	Résultat
1	Arbitrary Entailment	non traité
2	Chaining_strategy_numeric-add2	non traité
3	Chaining_strategy_numeric-subtract 1	non traité
4	Class Membership	réussi
5	Classification-inheritance	réussi
6	ElementEqualityFromListEquality	non traité
7	EntailEverything	non traité
8	Equality_in_conclusion_ 1	réussi
9	Equality_in_conclusion_2	réussi
10	Equality_in_conclusion_3	réussi
11	Equality_in_condition	non traité
12	Factorial_Functional	non traité
13	Factorial_Relational	réussi
14	Inconsistent_Entailment	non traité
15	Individual-Data_Separation_Inconsistency	non traité
16	IRI_from_IRI	non traité
17	ListConstantEquality	non traité
18	ListEqualityFromElementEquality	non traité

19	ListLiteralEquality	non traité
20	Multiple_IRIs_from_String	non traité
21	Multiple_Strings_From_IRI	non traité
22	Named_Arguments	réussi
23	RDFCombination Member 1	réussi
24	RDFCombination SubClas 4	réussi
25	RDFCombination SubClass 6	réussi
26	YoungParentDiscoun_1	réussi

TABLE 6.1: Résultats des tests pour l'implication positive

1	OpenLists	non traité
2	Classification_non-inheritance x	réussi
3	Named_Argument_Uniterms_non-polymorphic	réussi
4	OpenLists	non traité
5	RDF_Combination_SubClass_3	réussi
6	RDF_Combination_SubClass_5	réussi

TABLE 6.2: Résultats des tests pour l'implication négative

Tous les cas de test dont les fonctionnalités sont couvertes par le dialecte RIF-SPARQL ont réussi. Par comparaison avec RIF-BLD, RIF-SPARQL exclut les faits universels, les listes ouvertes et limite les termes de base à des constantes et des variables pour la construction de formules. Avec cette restriction, certaines formules de RIF-BLD n'ont pas de correspondant en SPARQL. Par conséquent, les cas test qui ne respectent pas cette restriction ne peuvent pas être traduits. L'explication détaillée des résultats de cette expérimentation est présentée dans l'annexe A.

6.3.5 Service RIF-SPARQL

Pour permettre aux utilisateurs de faire la validation et la traduction de leurs documents RIF-BLD en SPARQL, nous avons déployé sur le Web un service RIF2SPARQL [4] pour la validation de documents RIF-BLD et un traducteur des formules du dialecte RIF-SPARQL en SPARQL. Nous avons utilisé le modèle MVC pour développer le service en ligne. C'est ce que montre la Figure 6.8. L'application est développée en Java avec JSP/Servlet et les langages HTML et CSS. Elle est déployée avec le serveur Web Tomcat d'Apache.

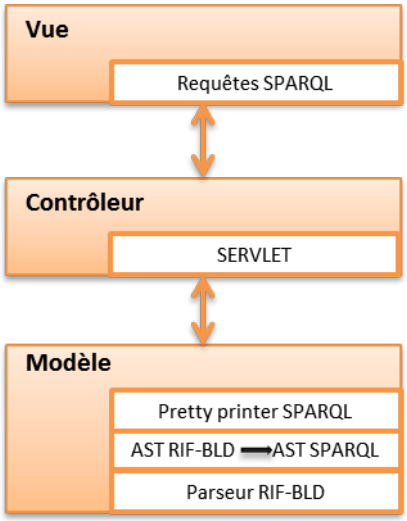

FIGURE 6.8 – Modèle de l'implémentation du service RIF2SPARQL

La Figure 6.9 montre l'interface de notre service de validation et de traduction des formules de RIF-BLD en SPARQL. On peut voir sur cette interface notamment la zone d'affichage du document RIF-BLD à traduire ainsi qu'une zone où sont affichés les résultats de la traduction de RIF-BLD en SPARQL, cette dernière zone est composée d'une sous-zone contenant les requêtes INSERT DATA résultant de la traduction des axiomes, une sous-zone contenant les requêtes CONSTRUCT résultant de la traduction des règles universelles et une sous-zone contenant le graphe

4. http://wimmics-ws.inria.fr/riftosparql/

RDF résultant de l'application des requêtes INSERT DATA issues de la traduction des axiomes avec Corese/KGRAM. Cette capture d'écran montre les résultats de la traduction du document RIF tiré de la base de test d'inférences positives sur RIF proposée par le W3C [5].

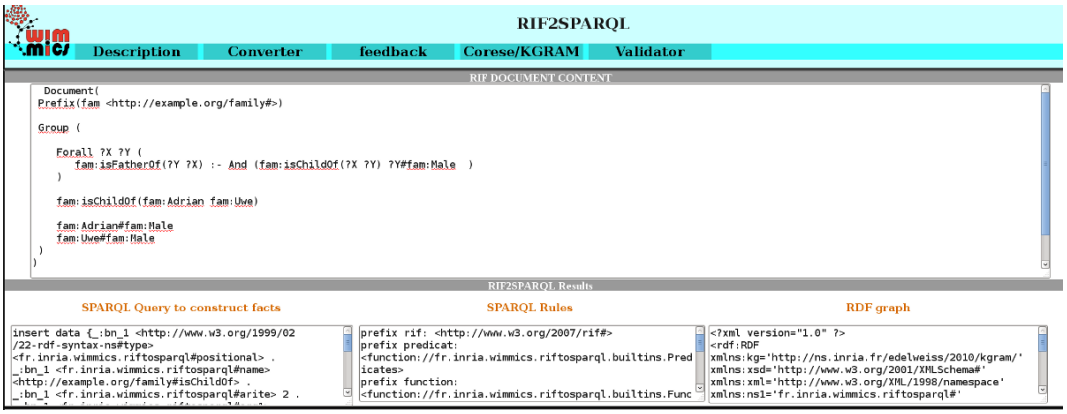

FIGURE 6.9 – Le service Web RIF2SPARQL : résultat de la traduction d'un document RIF-BLD

6.4 Expérimentation de la réutilisation de règles partagées

En combinant la sélection des règles représentées en RDF et leur application à des sources de données RDF, Corese/KGRAM permet de répondre à des scénarios complexes du Web de données, intégrant la sélection et l'application de règles sur des données du Web sans développement supplémentaire. Cette section présente nos implémentations de scénarios de sélection de règles en utilisant la plateforme Corese/KGRAM.

5. http://www.w3.org/2005/rules/test/repository/tc/Classification-inheritance/Classification-inheritance-premise.rifps

6.4.1 Preuve de concept

Nous avons mené des premières expériences sur les données du tutoriel de Corese/KGRAM. Comme les règles utilisées sont en syntaxe concrète de SPARQL, nous les avons traduites en un ensemble de 25 graphes SPIN/RDF avec le moteur de transformation SPIN de Corese/KGRAM [Corby 2014]. Nous avons sélectionné un *sous-ensemble* du schéma RDFS `human.rdfs` dans une source de données RDFS supplémentaire `human1.rdfs`. Nous avons alors utilisé Corese/KGRAM pour interroger les 25 règles SPIN/RDF et sélectionner les seules règles dont l'hypothèse fait référence à des ressources décrites dans le schéma `human1.rdfs`.

La requête SPARQL présentée dans le chapitre précédent sur la Figure 5.4 permet de sélectionner les sept règles SPIN souhaitées. Cette expérience montre la faisabilité de la sélection de règles en fonction de l'ontologie. En effet, certaines règles qui utilisent des concepts qui ne sont pas dans l'ontologie sont inutiles. Et donc, nous réduisons le coût de l'application de la base de règles en supprimant les règles qui ne s'appliquent pas à priori. Cette économie de temps peut ainsi devenir cruciale sur une grosse source de données.

6.4.2 Sélection de règles en mode distribué

Avec son méta-producteur, Corese/KGRAM peut utiliser ses moteurs de règles pour raisonner sur des données distribuées sur plusieurs sources. Il peut ainsi raisonner sur des données distribuées avec une base de règles SPARQL centralisée. De plus, comme nous représentons les règles en SPIN/RDF, la sélection de règles en amont de leur application peut également se faire dans un environnement distribué (Figure 6.10). Des sources de règles SPIN/RDF distribuées sont interrogées pour sélectionner des règles pertinentes et construire localement une base de règles à appliquer à des sources de données distribuées. Enfin, le choix des règles pourrait même impliquer l'application de méta-règles lors de la sélection des règles.

Pour expérimenter la réutilisation de règles en environnement distribué, nous

avons divisé les 25 règles SPIN/RDF dans deux sources distribuées. Nous avons interrogé les deux sources de règles SPIN/RDF distribuées pour sélectionner des règles pertinentes et construire une base de règles à appliquer à des sources de données distribuées. Comme illustré dans la Figure 6.10, pour cette expérience nous avons besoin de connecter le moteur de recherche KGRAM avec 3 producteurs RDF.

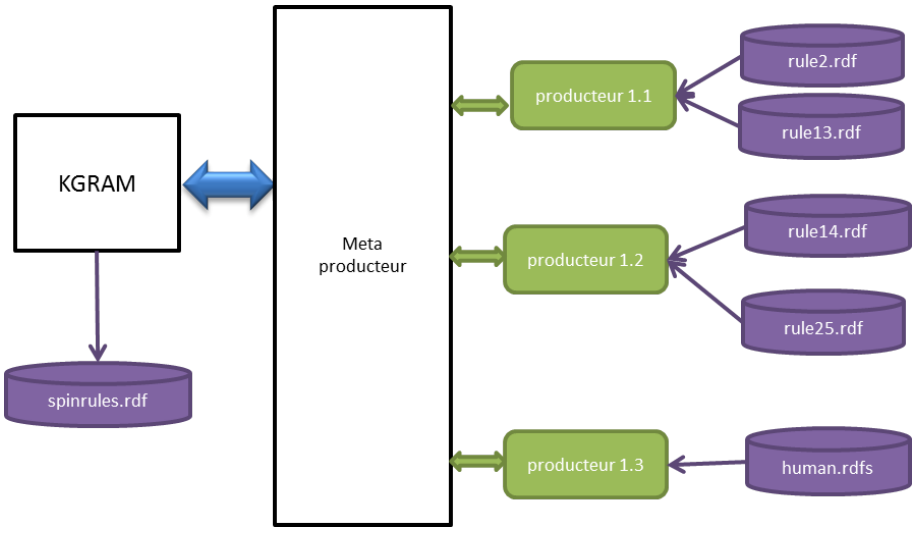

FIGURE 6.10 – Sélection de règles distribuées

6.4.3 Application de règles en distribué

Nous avons divisé les données RDF en deux sources de données RDF : `human1.rdf` et `human2.rdf`. Enfin, nous avons centralisé les 7 règles sélectionnées, étant donné le sous-ensemble de l'ontologie auquel nous nous limitons, et nous avons utilisé le moteur de règles de Corese/KGRAM pour appliquer ces règles sur les données RDF distribuées entre les sources `human1.rdf` et `human2.rdf`.

Cette expérience montre la faisabilité de l'application d'une base de règles centralisée sur des données distribuées. Cependant, l'application de règles d'inférence sur des sources de données distribuées peut s'avérer extrêmement coûteuse, lié au

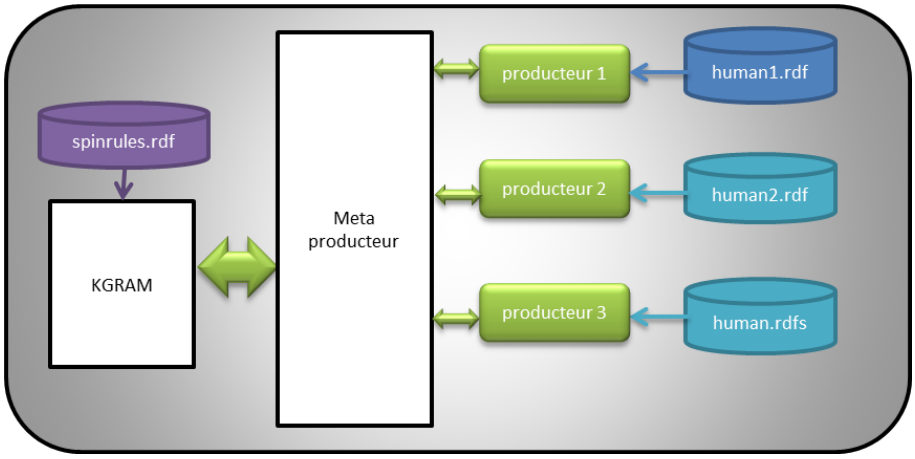

FIGURE 6.11 – Application de règles centralisées sur des données distribuées

coût d'échange réseau.

6.5 Optimisation du moteur d'inférence basée sur la sélection de règles transitives

Olivier Corby a effectué une optimisation du moteur de règles Corese/KGRAM basé sur la sélection de règles transitives, pour les traiter de manière particulière. L'optimisation consiste à exécuter à saturation les règles transitives dans une boucle dédiée. Cette optimisation a nettement amélioré la performance du moteur de règles de Corese/KGRAM comme le montre l'expérience suivante. En appliquant OWL RL Lite (OWL RL sans owl:sameAs) sur l'ontologie FMA [6] (Foundational Model of Anatomy) ontology, les temps de calcul suivants ont été obtenus :

- 126.253 sec avec optimisation transitive;
- 300.249 sec sans optimisation transitive.

Le graphe initial contient 1743162 triplets (1.74 million), le graphe obtenu après inférence contient 12514419 triplets (12.5 millions). Cette optimisation apporte donc un gain de temps de plus de 50% lors de l'application des règles OWL RL

6. http://sig.biostr.washington.edu/projects/fm/AboutFM.html

Lite sur l'ontologie FMA. Ceci montre la pertinence de la détection des règles transitives.

6.6 Optimisation du moteur d'inférence basée sur le calcul du graphe de dépendances entre règles

Nous avons utilisé le calcul du graphe de dépendances entre règles pour optimiser le moteur de règles de Corese/KGRAM.

6.6.1 Illustration sur un exemple

Soit `(r1, r2 ... r6)` la base des 6 règles d'inférence suivantes.

```
r1 :
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?x c:hasBrother ?z }
 WHERE {
   ?x c:hasParent ?y .
   ?z c:hasParent ?y .
   filter(?x != ?z)
   ?z rdf:type c:Man }

r2 :
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?x c:hasSister ?z }
 WHERE {
   ?x c:hasParent ?y .
   ?z c:hasParent ?y .
   filter(?x != ?z)
   ?z rdf:type c:Woman }
```

```
r3 :
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?x c:hasUncle ?z }
 WHERE {
   ?x c:hasParent ?y .
   ?y c:hasBrother ?z }

r4 :
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?x c:hasGrandParent ?z }
 WHERE {
   ?x c:hasParent ?y .
   ?y c:hasParent ?z }

r5:
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?x c:hasCousin ?t }
 WHERE {
   ?x c:hasParent ?y  .
   ?y c:hasParent ?a .
   ?t c:hasParent ?z .
   ?z c:hasParent ?a
   filter(?z != ?y)
   filter(?x != ?t) }

r6 :
 PREFIX c: <http://www.inria.fr/humans.rdfs#>
 CONSTRUCT {
   ?z c:hasRelation ?y }
 WHERE {
   ?z c:hasParent ?y }
```

Pour cette base de règles, il existe un seul lien de dépendance, entre `r3` et `r1`, avec la correspondance du motif de triplet `?x c:hasBrother ?z` de la clause CONSTRUCT de `r1` et le motif de triplet `?y c:hasBrother ?z` de la clause WHERE de `r3`. Avec un algorithme naïf, suite à l'activation de `r1` à une itération i, si `r1` infère de nouveaux résultats, toutes les règles de la base seront activées à l'itération i+1 tandis qu'en exploitant le graphe de dépendances seule la règle `r3` qui dépend de `r1` sera activée. Nous aurons ainsi au maximum deux itérations, quelles que soient les données sur lesquelles ces 6 règles seront appliquées et seule la règle `r3` sera activée à la deuxième itération à condition que l'application de la règle `r1` produise de nouveaux résultats à la première itération. Ce qui économiserait 6 activations inutiles de règles. Cette économie de la tentative de leur application devient précieuse dans le cas d'une source de données de grande taille.

6.6.2 Principe de l'optimisation

L'algorithme en chaînage avant naïf s'exécute de façon itérative tant que de nouveaux faits sont générés. Toutefois, pour deux règles R_i et R_j, si aucun modèle de triplet de la clause WHERE de R_i ne correspond à un modèle de triplet de la clause CONSTRUCT de R_j, alors l'application de R_j n'a aucun effet sur l'application de R_i. Par conséquent la réactivation de R_i à l'itération suivante est inutile.

Notre méthode d'optimisation du moteur d'inférences en chaînage avant du moteur de Corese/KGRAM est basée sur le calcul de graphe de dépendances de règles défini dans [Baget 2004], que nous avons adapté à des règles SPARQL. Dans cet algorithme toutes les règles de la base de connaissances sont exécutées à la première itération. Ensuite, à l'itération $k+1$, seules les règles qui dépendent des règles applicables (une règle est applicable si son application produit des résultats) à l'itération k seront déclenchées. Nous partons du graphe de dépendance entre règles de la base de connaissances, calculé sur les représentations SPIN des règles, avec une requête SPARQL. Nous avons aussi établi un ordre d'exécution des règles

du graphe de dépendance. Pour ce faire, chaque règle sera associée à un rang. L'affectation du rang se fait comme suit (cf Figure 6.12) :

- Le rang d'une règle par défaut est 0,
- Si une règle R_i dépend d'une règle R_j et `rang de` $R_i <$ `rang de` R_j alors remplacer le rang de R_i par `rang de` $R_j + 1$.

```
//mettre le rang de tous les sommets à 0
procedure initialiser(GrapheRegle rules)
 DEBUT
   pour tout sommet s de rules faire
     rang(s)=0;
   finFaire
 FIN

procedure ranger(GrapheRegle rules)
 DEBUT
   liste=initialiser(rules);
   pour tout sommet s de rules faire
     pour tout sommet w de rules faire
       si (s depend de w et rang(s) <= rang(w))
         rang(s) = rang(w) + 1
       finSi
       sinon
         si (w depend de s et rang(w) <= rang(s))
           rang(w) = rang(s) + 1
         finSi
       finSinon
     finFaire
   finFaire
 FIN
```

FIGURE 6.12 – Algorithme d'affection de rang

Nous avons ainsi défini un algorithme de parcours du graphe de dépendance par ordre de rang. Le parcours des sommets du graphe se fait par ordre de rang croissant, il consiste à explorer les arcs du graphe, en commençant par les sommets S de rang inférieur (c'est-à-dire la règles avec plus petit rang), puis tous les sommets plus proches de S, par rapport au rang, et ainsi de suite .... Par exemple, considérons le graphe de dépendance de règles de la Figure 6.13, les règles seront déclenchées dans l'ordre défini dans la Table 6.3 :

Pour parcourir le graphe par ordre de dépendance, les règles sont placées dans

sommet	r1	r5	r7	r3	r8	r6	r4	r2	r9
rang	0	1	1	2	2	3	3	4	4

TABLE 6.3 – Règles rangées par ordre de rang croissant

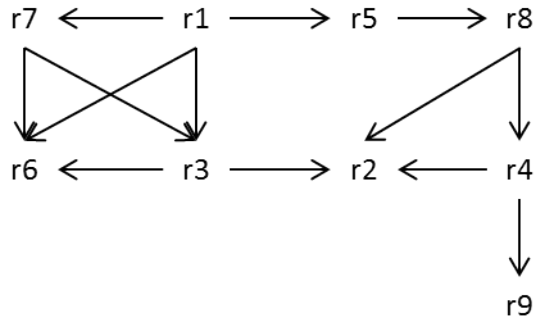

FIGURE 6.13 – Exemple de graphe de dépendances de règles

une file par rang croissant. Chaque règle est dotée d'un indicateur booléen actif. A la création de la base toutes les règles sont activées. Nous désactivons une règles suite à son application puis nous vérifions si son application a permis de produire de nouveaux faits, le cas échéant nous ajoutons et activons (mettre actif à vrai) toutes les règles inactives dépendantes d'elle sur la file d'attente d'exécution de règles. Le processus d'inférences s'arrête si aucune règle n'est activable. L'ordre induit par ce parcours permet de minimiser encore le nombre d'activations des règles. Par exemple, pour R_i et R_j deux règles avec R_i qui dépend de R_j, le fait d'appliquer R_j avant R_i permet de gagner une itération. Cette méthode réduit la complexité des inférences en réduisant le nombre d'activations des règles. Notre optimisation peut être résumée comme suit :

- Construction du graphe de dépendances entre règles de la base de connaissances,
- Si une règle R_i dépend d'une règle R_j alors déclencher R_j avant R_j dans la mesure du possible,
- Si une règle R_i dépend d'une règle R_j alors le déclenchement de R_i à l'itération $k+1$ sera conditionnée par l'application de R_j après R_i à l'itération

k.

6.6.3 Algorithme de chaînage avant optimisé

L'algorithme de chaînage avant optimisé que nous proposons est décrit sur la Figure 6.14. Il exploite le graphe de dépendances entre règles à travers l'affectation de rangs aux règles.

```
procedure inference(GrapheRegle rules)
DEBUT
  iteration = vrai;
  file = Ranger(rules); // ajout des règles sur la file d'attente
                        // par ordre de rang croissant
  tant que (iteration) faire
  iteration = faux;
  pour tout sommet s de rules faire
    entier i = executer(s);
    active(s) = faux; // désactivation de la règle
    si (i > 0) // génération de nouveaux triplets
      pour tout nœud w dépendant de s faire
        si (active(w) == faux )
          // règle déjà exécutée, toutes les règles sont activées au départ
          active(w) = vrai;
          iteration = vrai;
        finSi
      finFaire
    finSi
  finFaire
FIN
```

FIGURE 6.14 – Algorithme de chaînage avant optimisé

6.6.4 Évaluation

6.6.4.1 Expérimentation sur un jeu de données jouet

Nous avons évalué notre méthode en considérant le jeu de données utilisé dans la section 6.4.1, constituées d'une base de connaissances composée de 81 triplets du graphe RDF humain.rdf et 25 règles. A l'exécution du moteur en chaînage avant nous obtenons 4 itérations dont :

itération 1 : 25 règles activées

itération 2 : 12 règles activées

itération 3 : 5 règles activées

itération 4 : 1 règle activée.

En exploitant le graphe de dépendances entre règles, nous notons une baisse du nombre de règles activées à chaque itération. Au total cet algorithme a conduit à 43 exécutions de règles. Tandis que pour le moteur précédent basé sur l'algorithme naïf, l'exécution du moteur a abouti à l'activation de 100 règles.

6.6.4.2 Expérimentation avec la sémantique de OWL 2RL

L'axiomatisation de OWL 2 RL/RDF[7] est définie dans la recommandation du W3C. Elle est composée d'un ensemble de règles dont les antécédents et les conséquents sont sous forme de motifs de triplets RDF. Ces règles peuvent facilement être représentées avec des requêtes SPARQL de la forme CONSTRUCT. Par exemple la règle d'implication OWL 2 RL/RDF (prp-eqp2), qui énonce que si un graphe RDF contient les triplets `(P1 owl:equivalentProperty P2)` et `(X P2 Y)`, alors le triplet`(X P1 Y)` est inféré, s'écrit en règle SPARQL comme suit :

```
CONSTRUCT { ?x ?p1 ?y . }
WHERE {
  ?p1 owl:equivalentProperty ?p2 .
  ?x ?p2 ?y .
}
```

Dans ces expérimentations, nous nous appuyons sur la base des 71 règles SPARQL écrites dans OWL 2 RL in SPARQL[8] pour implémenter la sémantique de OWL 2 RL, le jeu de données RDF DBpedia-person, et le moteur de règles en chaînage avant de Corese/KGRAM. Nous avons mesuré le temps d'exécution de l'ensemble des règles par le moteur, avec et sans l'optimisation apportée avec le

7. http://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules

8. http://topbraid.org/spin/owlrl-all.html

graphe de dépendances, sur une machine dotée de 4Go de RAM et de deux CPUs Intel dual-core cadencés à 3GHz.

Nous appliquons les 71 règles de la sémantique de OWL 2 RL sur le graphe `persondata_en_uris_de.ttl` [9] contenant des données RDF décrivant la biographie des personnes possédant une page Wikipédia. Nous avons considéré un sous-ensemble de 200000 triplets, nous avons obtenu 856 772 triplets inférés (équivalent au nombre de triplets inférés avec le moteur de règles avant d'y apporter cette optimisation). A l'exécution du moteur en chaînage avant, nous obtenons 4 itérations dont les résultats sont présentés dans le tableau 6.4:

itérations	Avec optimisation	Sans optimisation
1	71 règles activées	71 règles activées
2	69 règles activées	71 règles activées
3	49 règles activées	71 règles activées
4	6 règle activées	71 règles activées

TABLE 6.4 – Réduction du nombre d'activations de règles OWL 2 RL en utilisant le graphe de dépendances entre règles

En exploitant le graphe de dépendances entre règles, nous notons une baisse du nombre de règles activées à chaque itération. Au total cet algorithme a conduit à 192 exécutions de règles. Tandis que pour le moteur précédent basé sur l'algorithme naïf, l'exécution du moteur a conduit à l'activation de 284 règles.

Le tableau 6.5 illustre le temps moyen d'application des règles sur cinq exécutions.

Données DBpedia-person	Sans optimisation	*Avec optimisation*
Nombre de triplets initiaux	200 000	200 000
Nombre de triplets initiaux et inférés	856 772	856 772
Temps moyen de calcul des inférences (s)	425,7	323,6
Ecart-type (s)	12	7.5

TABLE 6.5 – Réduction du temps de calcul des inférences OWL 2 RL avec l'utilisation de graphe de dépendances

9. http://downloads.dbpedia.org/3.9/de/persondata_en_uris_de.ttl.bz2

L'optimisation que nous avons apportée au moteur de règles permet de gagner plus de 47% d'activations de règles par rapport au moteur avant l'optimisation et apporte un gain de temps de plus de 25% lors de l'application des règles OWL 2 RL sur une partie les données DBpedia-person (200 000 triplets), et ce malgré la généricité de la base de règles OWL 2 RL. En effet, la base de règles OWL 2 RL met en jeu certaines règles dont la conclusion (clause CONSTRUCT) referme un motif de triplet composé de variables (?s ?p ?o), qui aboutit à beaucoup de liens de dépendances entre règles.

Cette expérimentation montre la pertinence de la prise en compte du graphe de dépendances entre règles dans le chaînage avant dans un contexte où les données peuvent être volumineuses.

6.7 Conclusion

Dans ce chapitre, nous avons présenté notre travail d'implémentation et d'évaluation. Nous avons effectué l'implémentation de RIF-SPARQL défini dans le chapitre 3 à l'aide d'un moteur SPARQL. Nous avons utilisé le parser RIF qui construit une représentation d'une règle RIF en un arbre de syntaxe abstraite (AST) et nous avons construit un traducteur du sous-arbre de cet AST qui correspond au dialecte RIF que nous avons défini en AST SPARQL de Corese/KGRAM, de sorte que celui-ci implémente le dialecte RIF-SPARQL que nous avons défini. Nous avons rendu ce traducteur accessible via un service Web (http://wimmics-ws.inria.fr/riftosparql/). Nous avons testé notre implémentation sur les cas de test publiés et approuvés par le groupe de travail RIF du W3C.

Ensuite, nous avons présenté et implémenté différents scénarios de réutilisation et de construction automatique de bases de règles pertinentes dans un contexte donné ; nous nous sommes basés sur l'interrogation de la description RDF des règles dans le langage SPARQL (SPIN).

Nous avons présenté une méthode d'optimisation de l'algorithme en chaînage

avant basé un graphe de dépendances entre règles, construit par interrogation de la description SPIN/RDF des règles. Nous avons implémenté cet algorithme dans le moteur Corese/KGRAM et nous avons conduit des expérimentations avec une base de règles implémentant la sémantique de OWL 2 RL, représentées en SPIN, et appliquée à des vocabulaires populaires du Web de données. Nous avons mesuré un gain en temps significatif que permet ce graphe de dépendances.

CHAPITRE 7

Conclusion

Bilan

Dans cette thèse, nous nous sommes intéressés au partage et à la réutilisation de règles sur le Web. Nous avons commencé par étudier RIF. Nous avons avons considéré la forme CONSTRUCT de SPARQL qui est utilisée comme langage de règles. Nous avons défini un dialecte de RIF que nous avons appelé RIF-SPARQL ; ce dialecte est le sous-ensemble de RIF-BLD traduisible en SPARQL.

Ensuite, nous nous sommes intéressés à une approche pour publier, partager et réutiliser des règles d'inférence de façon unifiée sur le Web de données liées. Nous avons choisi des règles SPARQL que nous exploitons dans leur syntaxe SPIN/RDF. Nous avons montré à travers l'utilisation du moteur sémantique Corese/KGRAM que la publication et la réutilisation de règles SPIN peuvent ainsi reposer de manière unifiée sur les modèles et techniques du Web de données. Nous nous sommes intéressés à la réutilisation de règles basée sur l'interrogation de leur description RDF dans le langage SPARQL. Nous avons présenté différents scénarios de construction automatique de bases de règles pertinentes dans un contexte donné à l'aide de requêtes SPARQL que nous avons écrites, il s'agit de construire pour une ontologie une base des règles pertinentes. Nous entendons par règles pertinentes celles susceptibles de s'appliquer sur les données et donc de produire de nouvelles données inférées. Il s'agit ici des règles contenant au moins un terme de l'ontologie. Lors de l'application de la base de règles ainsi créée à une source de données particulière, les techniques d'optimisation des raisonneurs basées sur l'ana-

lyse des dépendances de règles peuvent s'appliquer. Le moteur Corese/KGRAM permet d'implémenter les différents scénarios d'interrogation et d'application de règles multi-sources. Nous avons précisé plusieurs scénarios typiques de réutilisation de règles avec des données liées et nous les avons expérimentés avec un jeu de données de taille réduite. Cependant, l'application de règles d'inférence sur des sources de données distribuées peut s'avérer extrêmement couteuse.

La représentation de règles en SPIN/RDF permet de faire de la recherche de règles par le contenu et les métadonnées qui leur sont associées, ainsi que de la validation de base de règles avec l'utilisation de requêtes SPARQL Query. Cette représentation permet aussi la réécriture (modification) de règles avec l'utilisation de requêtes SPARQL Update. L'inconvénient de SPIN est sa syntaxe verbeuse, difficile à lire pour des utilisateurs humains, mais il existe des traducteurs SPIN vers SPARQL et SPARQL vers SPIN (e.g. Corese les implémente). Ainsi l'utilisateur peut interagir en syntaxe SPARQL avec le système qui calcule en format SPIN.

Enfin, nous avons adapté la technique d'optimisation des raisonneurs basées sur l'analyse des dépendances présentée dans [Baget 2004] au moteur de règles de Corese/KGRAM. Nous avons conduit des expérimentations d'optimisation du raisonneur optimisé du moteur de règles en chainage avant de de Corese/KGRAM sur les règles de sémantique de OWL 2 RL pertinentes pour le vocabulaire DBpedia-person et montré le gain en temps que permet une telle optimisation.

Notre approche et notre implémentation de partage et de réutilisation de règles se généralisent à tout langage de règles muni d'une syntaxe RDF ou qui peut être traduit dans le langage SPARQL. Notamment avec le dialecte RIF qui peut être traduit en SPARQL. Nous avons implémenté ce dialecte avec le moteur sémantique Corese/KGRAM et nous avons déployé un service en ligne pour la traduction des règles RIF-SPARQL en SPARQL. Ainsi, nous sommes capables de publier en RDF des règles RIF de ce dialecte en les traduisant d'abord dans le langage SPARQL puis en SPIN.

Perspectives

Une approche de réutilisation de règles similaire pourrait également être adoptée pour les langages de règles munis d'une syntaxe RDF qui permettrait de publier sur le Web de données les règles écrites dans ce langage. C'est le cas de SWRL [1], un langage de règles relativement utilisé bien que non standardisé (member Submission au W3C depuis 2004); cette méthode peut aussi être appliquée sur la syntaxe RDF de RIF [2] (Rule Interchange Format) la recommandation de W3C pour échanger des règles.

Nous pouvons exploiter les avancées sur les travaux liés à l'évolution d'ontologies pour détecter d'autres types d'incohérence d'une base de règles par rapport à une ontologie et y remédier avec l'utilisation de requêtes SPARQL. Nous pouvons envisager une interface pour présenter aux utilisateurs les incohérences détectées avec la proposition de correction.

Nous pouvons utiliser cette approche de publication et de réutilisation de règles pour construire un moteur de recherche de règles sur le Web, ou construire un outil permettant de relier automatiquement des règles avec des ressources du Web de données.

Il est aussi envisageable de travailler sur la traçabilité des règles lorsqu'elles sont sélectionnées sur le Web de données. Un travail de modélisation peut être conduit pour expliquer les résultats produits par la mise en œuvre d'inférences sur une base de règles. Il s'agit de modéliser la réussite ou l'échec et les explications des résultats produits, à partir de la trace du chaînage de règles mise en œuvre. Ce système de débogage permettrait de tracer la source des résultats et permettrait à l'utilisateur de comprendre les raisonnements effectués par le moteur, comment une réponse donnée a été obtenue et de spécifier l'origine de ses résultats et la possibilité de supprimer une ou plusieurs règles avec une fonctionnalité de mise à

1. http://www.w3.org/Submission/SWRL/
2. http://www.w3.org/TR/rif-in-rdf/

jour.

ANNEXE A

Explication des résultats de l'évaluation du dialecte RIF-SPARQL

Dans cette annexe nous expliquons dans le détail pourquoi certains cas de test n'ont pas été traités.

- Le non traitement de certains cas de test est lié à l'utilisation de termes de base non admis en RIF-SPARQL. Les termes de base de RIF-SPARQL sont limités aux constantes et aux variables. En effet, en RIF-SPARQL, seuls les termes simples sont admis dans la construction de termes et formules de RIF-SPARQL. Les externes, les termes de position et les listes sont exclus dans les termes de base de RIF-SPARQL. D'où l'échec de la traduction des cas de test suivants:
  - cas de test numéro 2 Chaining_strategy_numeric-add2 [1] et le cas de test numéro 3 Chaining_strategy_numeric-subtract [2] : ces deux cas de test comportent un terme externe dans la construction d'un terme contenu dans leur prémisse.
  - cas de test numéro 6, ElementEqualityFromListEquality [3] : dans ce cas de test, la prémisse inclut une égalité entre deux listes.

    `Group(`

1. http://www.w3.org/2005/rules/wiki/Chaining_strategy_numeric-add_2
2. http://www.w3.org/2005/rules/wiki/Chaining_strategy_numeric-subtract_1
3. http://www.w3.org/2005/rules/wiki/ElementEqualityFromListEquality

```
List(ex:a) = List(ex:b)
)
```

– cas de test numéro 11 (Equality_in_condition [4]) et cas de test numéro 12 (Factorial_Functional [5]): dans ces cas de test le document prémisse renferme un terme utilisant dans sa construction un externe, alors qu'en RIF-SPARQL les termes de base sont limités aux variables et aux constantes.

```
 External(func:numericsubtract(
   External(func:numeric-multiply(
               ?x
               External(func:numeric-multiply(?x ?x))))
   ?x))
= "0"^^xs:decimal)
```

– cas de test numéro 17, ListConstantEquality [6] : ce cas de test comporte une égalité entre une liste et une constante.

```
 ex:a = List(ex:b ex:c)
```

– cas de test numéro 18, ListEqualityFromElementEquality [7] : Ce cas de test comporte un terme d'égalité entre listes.

```
List(ex:a) = List(ex:b)
```

– cas de test numéro 19, ListLiteralEquality [8] : ce cas de test comporte un terme d'égalité dont l'un des arguments est une liste.

```
Group(
   "a" = List(ex:b)
  )
```

4. http://www.w3.org/2005/rules/wiki/Equality_in_condition
5. http://www.w3.org/2005/rules/wiki/Factorial_Functional
6. http://www.w3.org/2005/rules/wiki/ListConstantEquality
7. http://www.w3.org/2005/rules/wiki/ListEqualityFromElementEquality
8. http://www.w3.org/2005/rules/wiki/ListLiteralEquality

– Le non traitement de formules de certains cas de test est lié à l'exclusion d'un sous ensemble de terme d'égalité de RIF-BLD par RIF-SPARQL. En effet, les termes d'égalité ne peuvent être utilisés que sur les conditions des règles universelles en RIF-SPARQL. Cette restriction conduit à l'échec de la traduction des cas de test numéro 1(Arbitrary Entailment [9]), numéro 16 (IRI_from_IRI [10]), numéro 20 (Multiple_IRIs_from_String [11]) et numéro 21 (Multiple_Strings_From_IRI [12]). Dans ces cas de test, le document de prémisse contient un terme d'égalité dans leur axiome tandis que les termes d'égalité sont autorisés que dans les conditions de règles universelles de RIF-SPARQL.

```
Group( "a" = "b")
```

– Le non traitement du cas de test numéro 15 (Individual-Data_Separation_Inconsistency) [13] est lié au fait que le dialecte RIF-SPARQL exclut les faits universels.

```
Group (
 Forall ?x (?x[rdf:type -> ex:A])
  )
```

– Le non traitement du cas de test 30 (OpenLists) [14] s'explique par l'exclusion des listes ouvertes par RIF-SPARQL.

```
Group(
ex:p(List(ex:a | "b"))
 )
```

9. http://www.w3.org/2005/rules/wiki/Arbitrary_Entailment
10. http://www.w3.org/2005/rules/wiki/IRI_from_IRI
11. http://www.w3.org/2005/rules/wiki/Multiple_IRIs_from_String
12. http://www.w3.org/2005/rules/wiki/Multiple_Strings_from_IRI
13. http://www.w3.org/2005/rules/wiki/Individual-Data_Separation_Inconsistency
14. http://www.w3.org/2005/rules/wiki/OpenLists

Bibliographie

[Angles 2008] Renzo Angles et Claudio Gutierrez. *The Expressive Power of SPARQL*. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin et Krishnaprasad Thirunarayan, editeurs, The Semantic Web - ISWC 2008, volume 5318 of *Lecture Notes in Computer Science*, pages 114–129. Springer Berlin / Heidelberg, 2008. http://dx.doi.org/10.1007/978-3-540-88564-1_8. (Cité en pages 35, 54 et 61.)

[Baader 2005] Franz Baader, Sebastian Brand et Carsten Lutz. *Pushing the EL envelope*. In Proceedings of IJCAI 2005, pages 364–369. Morgan-Kaufmann Publishers, 2005. (Cité en page 7.)

[Baget 2004] Jean-François Baget. *Improving the Forward Chaining Algorithm for Conceptual Graphs Rules*. In Proc. 9th international conference on principles of knowledge representation and reasoning (KR), Proc. 9th international conference on principles of knowledge representation and reasoning (KR), pages 407–414, Whistler, Canada, Juin 2004. No commercial editor. baget2004b. (Cité en pages 73, 78, 110 et 120.)

[Baget 2009] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Eric Salvat*et al.* *Extending Decidable Cases for Rules with Existential Variables*. In IJCAI, volume 9, pages 677–682, 2009. (Cité en page 74.)

[Berners-Lee 2008a] Tim Berners-Lee et Dan Connolly. *Notation3 (N3): A readable RDF syntax*. W3C submission, W3C, January 2008. http://www.w3.org/TeamSubmission/n3/. (Cité en page 31.)

[Berners-Lee 2008b] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf et Jim Hendler. *N3Logic: A logical framework for the World Wide Web*. Theory and Practice of Logic Programming, vol. 8, pages 249–269, 4 2008. (Cité en page 49.)

[Bizer 2009] Christian Bizer, Tom Heath et Tim Berners-Lee. *Linked Data - The Story So Far*. International Journal on Semantic Web and Information Systems (IJSWIS), 2009. (Cité en page 21.)

[Bizer 2010] Christian Bizer et Andreas Schultz. *The R2R framework: Publishing and Discovering Mappings on the Web*. In Proceedings of the 1st Int. Workshop on Consuming Linked Data, COLD 2010, Shanghai, China. CEUR-WS.org, 2010. (Cité en page 35.)

[Boley 2001] Harold Boley, Said Tabet et Gerd Wagner. *Design Rationale for RuleML : A Markup Language for Semantic Web Rules*. SWWS, pages 381–401, 2001. (Cité en pages 31 et 32.)

[Boley 2007a] Harold Boley. *Are Your Rules Online? Four Web Rule Essentials*. In Adrian Paschke et Yevgen Biletskiy, editeurs, Advances in Rule Interchange and Applications, volume 4824 of *Lecture Notes in Computer Science*, pages 7–24. Springer Berlin Heidelberg, 2007. (Cité en page 46.)

[Boley 2007b] Harold Boley, Michael Kifer, Paula-Lavinia Pătrânjan et Axel Polleres. *Rule Interchange on the Web*. In Grigoris Antoniou, Uwe Aßmann, Cristina Baroglio, Stefan Decker, Nicola Henze, Paula-Lavinia Patranjan et Robert Tolksdorf, editeurs, Reasoning Web, volume 4636 of *Lecture Notes in Computer Science*, pages 269–309. Springer Berlin Heidelberg, 2007. (Cité en page 46.)

[Boley 2010a] Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke Axel Polleres et Dave Reynolds. *RIF Core Dialect*. W3C Recommendation, W3C, June 2010. http://www.w3.org/TR/2010/REC-rif-core-20100622/. (Cité en page 39.)

[Boley 2010b] Harold Boley et Michael Kifer. *A Guide to the Basic Logic Dialect for Rule Interchange on the Web*. the IEEE Computer Society, pages 1593–1608, November 2010. (Cité en page 44.)

[Boley 2010c] Harold Boley et Michael Kifer. *RIF Basic Logic Dialect*. W3C Recommendation, W3C, June 2010. http://www.w3.org/TR/2010/REC-rif-bld-20100622/. (Cité en page 39.)

[Boley 2010d] Harold Boley et Michael Kifer. *RIF Framework for Logic Dialects*. W3C Recommendation, W3C, June 2010. http://www.w3.org/TR/rif-fld/. (Cité en page 42.)

[Brachman 1984] Ronald J Brachman et Hector J Levesque. *The Tractability of Subsumption in Frame-Based Description Languages*. In Proceedings of AAAI 1984, pages 34–37, 1984. (Cité en page 7.)

[Calvanese 2007] Diego Calvanese, Giuseppe Giacomo, Domenico Lembo, Maurizio Lenzerini et Riccardo Rosati. *Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family*. Journal of Automated Reasoning, vol. 39, no. 3, pages 385–429, 2007. (Cité en page 27.)

[Corby 2004] Olivier Corby, Rose Dieng-kuntz et Catherine Faron-zucker. *Querying the Semantic Web with the CORESE search engine*. In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), volume 16, pages 705–709. IOS Press, 2004. (Cité en pages 91 et 92.)

[Corby 2010] Olivier Corby et Catherine Faron-Zucker. *The KGRAM Abstract Machine for knowledge graph Query*. In International Conference on Web Intelligence, Toronto, Canada, September 2010. (Cité en page 91.)

[Corby 2012] Olivier Corby, Alban Gaignard, Catherine Faron-Zucker et Johan Montagnat. *KGRAM Versatile Data Graphs Querying and Inference Engine*. In Proceedings IEEE/WIC/ACM International Conference on Web Intelligence, Macau, China, December 2012. (Cité en pages 91 et 92.)

[Corby 2014] Olivier Corby et Catherine Faron-Zucker. *SPARQL Template : un langage de Pretty Printing pour RDF*. In Catherine Faron-Zucker, editeur, IC - 25èmes Journées francophones d'Ingénierie des Connaissances, pages

213–224, Clermont-Ferrand, France, Mai 2014. Session 4 : Web sémantique. (Non cité.) :hal-0101526791, 94 et 105391, 94 et 105

[d'Aquin 2012] Mathieu d'Aquin et Natalya F. Noy. *Where to publish and find ontologies? A survey of ontology libraries*. Web Semantics: Science, Services and Agents on the World Wide Web, vol. 11, no. 0, pages 96 – 111, 2012. (Cité en page 45.)

[de Bruijn 2010] Jos de Bruijn. *RIF RDF and OWL Compatibility*. W3C Recommendation, W3C, June 2010. http://www.w3.org/TR/2010/REC-rif-rdf-owl-20100622/. (Cité en pages 39, 43, 58, 62 et 100.)

[Forgy 1982] Charles L. Forgy. *Rete: A fast algorithm for the many pattern/many object pattern match problem*. Artificial Intelligence, vol. 19, no. 1, pages 17 – 37, 1982. (Cité en page 78.)

[González-Moriyón 2012] Guillermo González-Moriyón, Luis Polo, Diego Berrueta, Carlos Tejo-Alonso et Miguel Iglesias. *Assembling Rule Mashups in the Semantic Web*. In 9th Extended Semantic Web Conference (ESWC2012), May 2012. (Cité en pages 50 et 67.)

[Grimm 2007] Stephan Grimm, Uwe Keller, Holger Lausen et Gábor Nagypál. *A Reasoning Framework for Rule-Based WSML*. In Enrico Franconi, Michael Kifer et Wolfgang May, editeurs, The Semantic Web: Research and Applications, volume 4519 of *Lecture Notes in Computer Science*, pages 114–128. Springer Berlin / Heidelberg, 2007. http://dx.doi.org/10.1007/978-3-540-72667-8_10. (Cité en page 58.)

[Grosof 2009] Benjamin Grosof, Mike Dean et Michael Kifer. *The SILK System: Scalable and Expressive Semantic Rules*. In 8th International Semantic Web Conference (ISWC2009), October 2009. `http://data.semanticWeb.org/conference/iswc/2009/paper/poster_demo/175`. (Cité en pages 34 et 44.)

[Gruber 1993] Thomas R. Gruber. *A translation approach to portable ontology specifications*. Knowledge aquisition, vol. 5, pages 199–220, 1993. (Cité en page 6.)

[Harris 2013] Steve Harris et Andy Seaborne. *SPARQL 1.1 Query Language*. Rapport technique, W3C, march 2013. (Cité en page 92.)

[Hawke 2013] Sandro Hawke. *RIF in RDF*. W3C Working Groupe Note, W3C, February 2013. http://www.w3.org/TR/2013/NOTE-rif-in-rdf-20130205/. (Cité en pages 39 et 42.)

[Hayes 2014a] Patrick Hayes. *RDF Semantics*. W3C Recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/. (Cité en page 43.)

[Hayes 2014b] Patrick J. Hayes et Peter F. Patel-Schneider. *RDF Semantics*. W3C Recommendation, W3C, February 2014. http://www.w3.org/TR/rdf11-mt/. (Cité en pages 23 et 75.)

[Horrocks 2004] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof et Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C member submission, W3C, May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/. (Cité en pages 31 et 33.)

[Khandelwal 2011] Ankesh Khandelwal, Ian Jacobi et Lalana Kagal. *Linked Rules : Principles for Rule Reuse on the Web*. In Web reasoning and Rule Systems, pages 108–123, 2011. (Cité en pages 46 et 48.)

[Kifer 1995] Michael Kifer, Georg Lausen et James Wu. *Logical foundations of object-oriented and frame-based languages*. Journal of the ACM (JACM), vol. 42, no. 4, pages 741–843, 1995. (Cité en page 32.)

[Lee 2003] Jae Kyu Lee et Mye M. Sohn. *The eXtensible Rule Markup Language*. Commun. ACM, vol. 46, pages 59–64, May 2003. http://doi.acm.org/10.1145/769800.769802. (Cité en page 31.)

[Marte 2011] Adrain Marte. RIF4J - A reasoning Engine for RIF-BLD. Master's thesis, University of Innsbruck, 2011. (Cité en page 58.)

[Mugnier 1996] Marie-Laure Mugnier et Michel Chein. *Représenter des connaissances et raisonner avec des graphes*. Revue d'intelligence artificielle, vol. 10, no. 1, pages 7–56, 1996. (Cité en page 9.)

[Neches 1991] Robert Neches, Richard E. Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator et William R. Swartout. *Enabling technology for knowledge sharing*. AI magazine, vol. 12, no. 3, page 36, 1991. (Cité en page 6.)

[Pan 2005] Jeff Pan, Giorgos Stamou, Vassilis Tzouvaras et Ian Horrocks. *f-SWRL: A Fuzzy Extension of SWRL*. In Wlodzislaw Duch, Janusz Kacprzyk, Erkki Oja et Slawomir Zadrozny, editeurs, Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005, volume 3697 of *Lecture Notes in Computer Science*, pages 752–752. Springer Berlin / Heidelberg, 2005. (Cité en page 44.)

[Polleres 2007a] Axel Polleres. *From SPARQL to Rules (and Back)*. In Proceedings of the 16th International Conference on World Wide Web, WWW '07, pages 787–796, New York, NY, USA, 2007. ACM. (Cité en pages 35 et 54.)

[Polleres 2007b] Axel Polleres, François Scharffe et Roman Schindlauer. *SPARQL++ for mapping between RDF vocabularies*. In Proceedings of the 2007 OTM Confederated international conference on the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I, OTM'07, pages 878–896, Berlin, Heidelberg, 2007. Springer-Verlag. http://portal.acm.org/citation.cfm?id=1784607.1784685. (Cité en page 35.)

[Polleres 2010] Axel Polleres, Harold Boley et Michael Kifer. *RIF Datatypes and Built-Ins 1.0*. W3C Recommendation, W3C, June 2010. http://www.w3.

org/TR/2010/REC-rif-dtb-20100622/. (Cité en page 39.)

[Schenk 2008] Simon Schenk et Steffen Staab. *Networked Graphs: a Declarative Mechanism for SPARQL rules, SPARQL Views and RDF Data Integration on the Web*. In Proceedings of the 17th Int. Conference on World Wide Web, WWW 2008, Beijing, China, pages 585–594. ACM, 2008. (Cité en page 35.)

[Seye 2012a] Oumy Seye, Catherine Faron-Zucker, Olivier Corby et Corentin Follenfant. *Bridging the Gap between RIF and SPARQL*. In Artificial Intelligence meets the Web of Data, ECAI Workshop, Montpellier, August 2012. (Cité en pages 3 et 53.)

[Seye 2012b] Oumy Seye, Catherine Faron-Zucker, Olivier Corby et Corentin Follenfant. *Définition et implémentation d'un dialecte RIF à l'aide d'un moteur SPARQL*. In 12ème Conférence Internationale Francophone sur l'Extraction et la Gestion de Connaissance, Session Poster, Bordeaux, Février 2012. (Cité en pages 3 et 53.)

[Seye 2014] Oumy Seye, Catherine Faron-Zucker, Olivier Corby et Alban Gaignard. *Publication, partage et réutilisation de règles sur le Web de données*. In Catherine Faron-Zucker, editeur, IC - 25èmes Journées francophones d'Ingénierie des Connaissances, pages 237–248, Clermont-Ferrand, France, Mai 2014. Session 4 : Web sémantique. (Cité en pages 3 et 68.)

[Sintek 2002] Michael Sintek et Stefan Decker. *TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web*. In Proceedings of the First International Semantic Web Conference, ISWC '02, pages 364–378, London, United Kingdom, 2002. Springer-Verlag. http://portal.acm.org/citation.cfm?id=646996.711416. (Cité en pages 31 et 32.)

[Sowa 1984] J. Sowa. Conceptual Structures. Addison-Wesley, 1984. (Cité en page 8.)

[Stoilos 2005] Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras et Jeff Pan. *Uncertainty and RuleML Rulebases: A Preliminary Report*. In Asaf Adi, Suzette Stoutenburg et Said Tabet, editeurs, Rules and Rule Markup Languages for the Semantic Web, volume 3791 of *Lecture Notes in Computer Science*, pages 199–203. Springer Berlin / Heidelberg, 2005. (Cité en page 44.)

[W3C 2005] W3C. *Rule interchange Format Working Group*. W3C Recommendation, W3C, 2005. http://www.w3.org/2005/rules/wiki/RIF_Working_Group. (Cité en page 100.)

[Wagner 2005] Gerd Wagner, Adrian Giurca et Sergey Lukichev. *R2ML: A general approach for marking up rules*. In Principles and Practices of Semantic Web Reasoning, Dagstuhl Seminar Proceedings 05371. Ohlbach (Eds), 2005. (Cité en page 47.)

[Wang 2008] Xing Wang, Z. Ma, Li Yan et Xiangfu Meng. *Vague-SWRL: A Fuzzy Extension of SWRL*. In Diego Calvanese et Georg Lausen, editeurs, Web Reasoning and Rule Systems, volume 5341 of *Lecture Notes in Computer Science*, pages 232–233. Springer Berlin / Heidelberg, 2008. (Cité en page 44.)

[Wang 2010] Xing Wang, Zongmin Ma, Li Yan et Runan Zhao. *RIF-FRD: A RIF dialect based on fuzzy sets*. In Fuzzy Systems and Knowledge Discovery (FSKD), volume 4, pages 1912 –1916, August 2010. (Cité en page 44.)

[Yang 2003] Guizhen Yang, Michael Kifer et Chang Zhao. *FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web*. In ODBASE 2003, November 2003. (Cité en page 31.)

[Zacharias 2008] Valentin Zacharias. *Tool Support for Finding and Preventing Faults in Rule Bases*. PhD thesis, Universitat Karlsruhe, 2008. (Cité en page 80.)

[Zadeh 1965] Lofti Zadeh. *Fuzzy sets*. Information and Control, vol. 8, no. 3, pages 338 – 353, 1965. http://www.sciencedirect.com/science/article/B7MFM-4DX43MN-W3/2/f244f7a33f31015e819042700cd83047. (Cité en page 44.)